

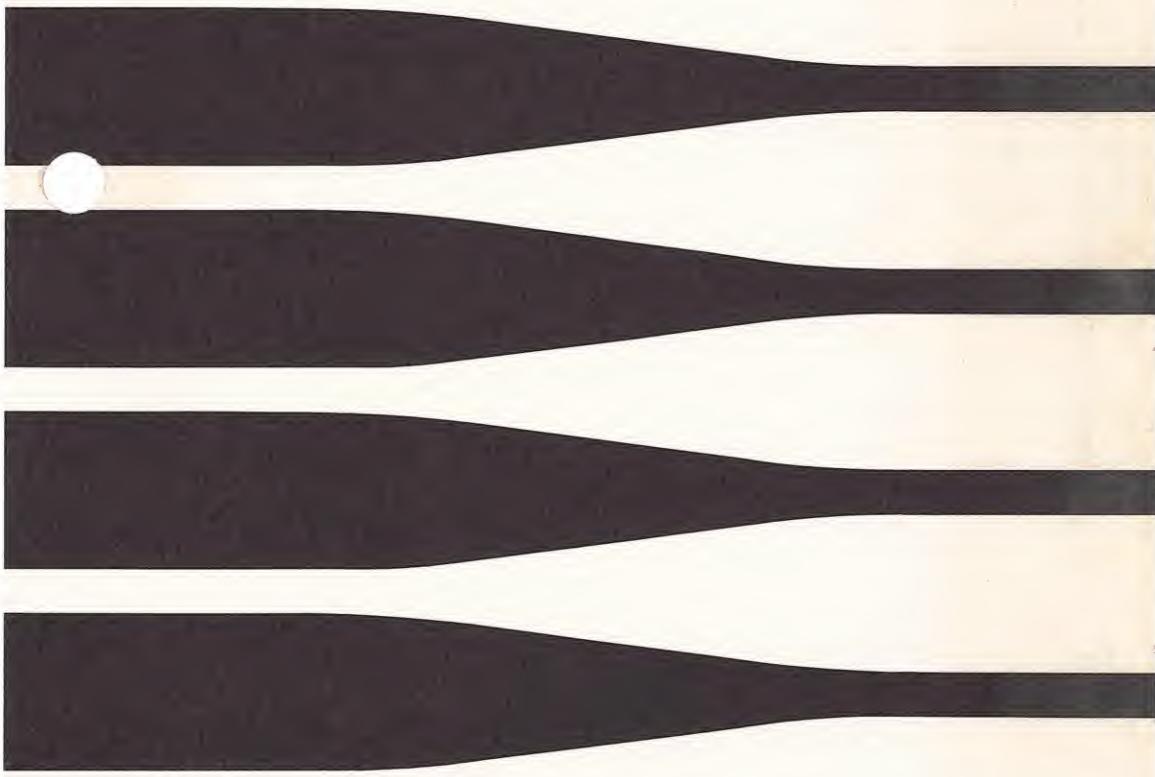


Issue 1
October 1984

AT&T 3B2 Computer UNIX™ System V Release 2.0 User Reference Manual

Select Code
305-406

Comcode
403778319



A decorative graphic consisting of four thick, dark horizontal bands that curve upwards towards the right. Between these bands are three thin, light-colored horizontal bands. A single white circular hole punch is visible on the left edge of the page, corresponding to the top band.

Copyright © 1984 AT&T Technologies, Inc.
All Rights Reserved
Printed in U.S.A.

DEC and PDP are trademarks of
Digital Equipment Corporation.

HP is a trademark of Hewlett-Packard, Inc.

DIABLO is a registered trademark of Xerox Corporation.

TEKTRONIX is a registered trademark of Tektronic, Inc.

Versatec is a registered trademark of Versatec Corporation.

TELETYPE is a trademark of AT&T Teletype Corporation.

DOCUMENTER'S WORKBENCH is a trademark of AT&T Technologies.

UNIX is a trademark of AT&T Bell Laboratories.

*This manual was set on an AUTOLOGIC, Inc.
APS-5 phototypesetter driven by the TROFF
formatter operating under the UNIX system.*

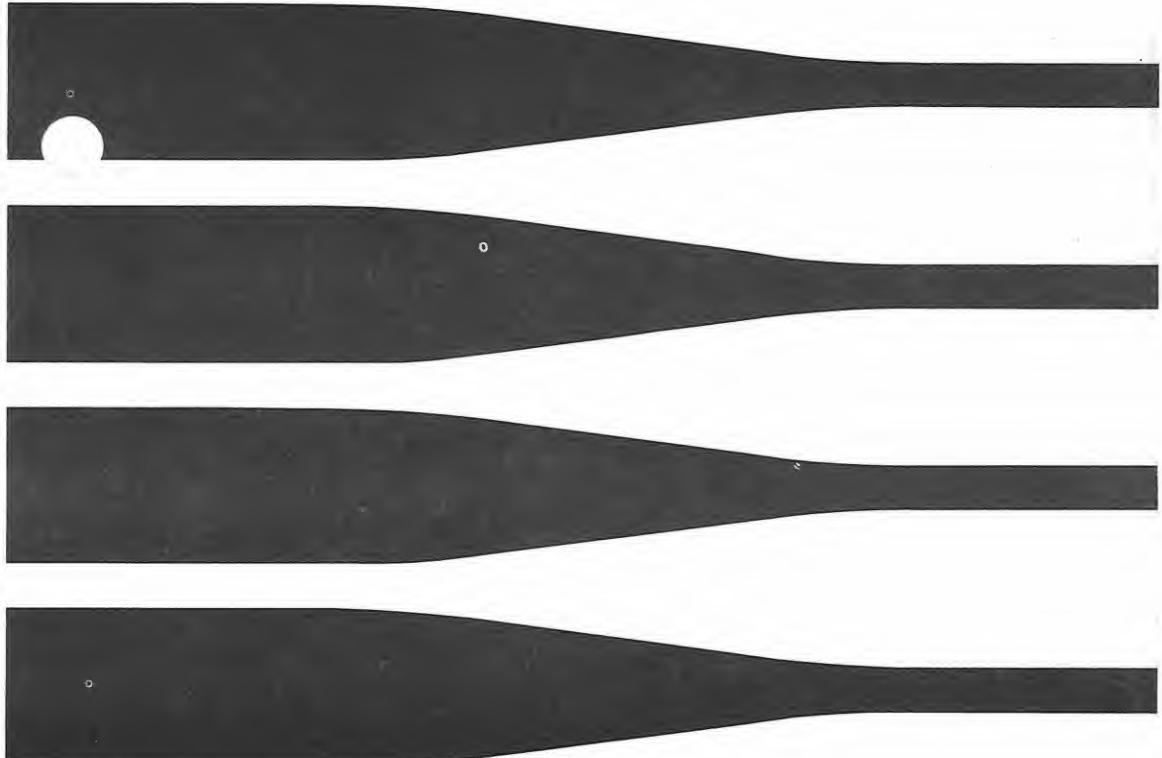


Issue 2
December 1984

**AT&T 3B2 Computer
UNIX™ System V Release 2.0
System Administration Utilities
Software Information Bulletin**

Select Code
305-359

Comcode
403778202



Copyright © 1984 AT&T
All Rights Reserved
Printed in U.S.A.

NOTE

This Software Information Bulletin (SIB) should be filed in the *3B2 Computer Owner/Operator Manual*. A tab separator, labeled " SOFTWARE INFORMATION BULLETINS," has been placed at the back of the *Owner/Operator Manual* in order to provide a convenient place for filing SIB's. Place the tab separator provided with this SIB in front of the title page and file this material behind the SOFTWARE INFORMATION BULLETINS tab separator in the *Owner/Operator Manual*.

SYSTEM ADMINISTRATION SOFTWARE INFORMATION BULLETIN

INTRODUCTION

This Software Information Bulletin provides important information concerning the System Administration Utilities. Please read this bulletin carefully before attempting to install or use these utilities.

The AT&T 3B2 Computer System Administration Utilities are for use by a sophisticated user, who needs more system administration capabilities than those provided by Simple Administration. The System Administration Utilities are part of the UNIX* System V Release 2.0 configuration provided with all 3B2 Computers.

SOFTWARE DEPENDENCIES

The System Administration Utilities are independent of all optional utilities. The System Administration Utilities are used with certain other system administration-type commands that are provided as part of the Essential Utilities. The Essential Utilities are already installed on the integral hard disk when you receive the machine.

* Trademark of AT&T Bell Laboratories

NOTES ON USING UTILITIES

Command Access

To use most of the system administration-type commands, you must be logged in on the system as **root**. Many system administration commands require the user to be a member of the "system administration group" (sys or adm). Thus, owner and group access permissions are associated with the following names: **root**, **sys**, **adm**, and **bin**.

The majority of the system administration-type commands are in the **/etc** directory. The **/etc** directory is in the normal search path of **root**; this directory is not in the default search path for normal user logins. Certain commands in the **/etc** directory are executable by a normal user providing that the PATH variable includes the **/etc** directory or that the complete path name is specified for the command.

Update Information

This section contains important update information about your 3B2 Computer and its Operating System. This information is categorized according to function, command category, etc. Read this information carefully.

Diagnostics Exceptions - System Board

Aborting phase 20 may take up to 6 minutes for a response.

Aborting system board diagnostics when undergoing a soak of phase 20, may take up to 6 minutes to receive a response from the system.

Backup/Restore/Upgrade Exceptions

1. **Multi-disk backups are invalid if a bad floppy diskette sector is encountered.** During a *sysadm backup*, if a bad sector is encountered on a floppy diskette, the user is not made aware of the problem. Further, when a *sysadm restore* is executed, the data from the point of the bad sector throughout the remaining backup diskettes is unreadable. Frequent individual file or directory backups are recommended in conjunction with periodic complete backups. The file and directory backup commands are available in the *store* menu under the *filemgmt* menu of simple administration.

2. **Misleading error messages are reported during the restore procedure.** During a "complete" restore of / and /usr, the following messages will be displayed:

```
Cannot link </bin/sh> & </bin/rsh>
Cannot link </etc/init> & </etc/telinit>
Cannot unlink current </etc/cron> error 26 text file busy
Cannot unlink current </etc/getty> error 26
Cannot create </etc/getty> error 26 text file busy
Cannot unlink current </etc/hdelogger> error 26
Cannot create </etc/hdelogger> error 26
Cannot unlink current </usr/lbin/ncpio> error 26
Cannot create </usr/lbin/ncpio> error 26
```

This is the normal operation of restore and does not indicate a restore problem or failure.

3. **Backup of root filesystem does not include " /dev " files.** When a backup of the *root* filesystem is performed, all directories and all files are copied, except the */dev* directory. If you wish to save these files, it is recommended that the following commands be issued:

```
# sysadm mountfsys<CR> Note: assuming a blank, formatted
floppy which has a filesystem
on it.
# find /dev -print | cpio -pd1 /filesys-name<CR>
# sysadm umountfsys<CR>
```

When the files are to be restored, execute the following set of commands:

```
# sysadm mountfsys<CR>  Note: after inserting the
                           floppy on which the
                           files were saved.
# cd filesys-name<CR>
# find /dev -print | cpio -pd1 /dev<CR>
# sysadm umountfsys<CR>
```

4. **"Upgraded" systems are slightly different than "restored" systems.** Systems which are upgraded to UNIX System V Release 2.0 will have slightly less swap space than newly purchased systems or those which have been fully restored to Release 2.0. For a 30-megabyte disk, an upgraded system will have 5120 blocks of swap space, and a restored system will have 6020 blocks of swap space. The amount of swap space on a 10-megabyte disk will not change (3500 blocks). The difference is minimal on the 30-megabyte disk; but if you are experiencing swap problems, it is suggested that a complete backup of user files and a full restore be performed. You also have the option to increase the size of the swap space partition during the full restore. See the **prtvtoc(1M)** command.

Initialization-Shutdown Exceptions

NVRAM sanity failure causes the floppy drive to become the default boot device. If an NVRAM Sanity Failure occurs, and the 3B2 Computer is immediately powered down, the default boot device will be changed to the floppy drive and all subsequent boot attempts will fail. To reset the default boot device back to the hard disk, perform the following procedures:

1. Power down the system.
2. Insert the floppy key into the floppy drive.
3. Power up the system. This will automatically execute the floppy key and reset NVRAM to default values. The system will then automatically boot the UNIX System from the hard disk.

Single-User Mode Exceptions

1. **The system hangs when "ctrl d" is entered in single user mode.** Before logging off in single user mode, (via <ctrl d>), you should return to the multi-user state by executing the following command:

```
# init 2<CR>
```

and then log off of the system.

If however, a *ctrl d* is entered in single user mode, the terminal baud rate can be changed to 300 baud, and you will regain control of the console terminal.

2. **The power switch is not operational in single user mode.** In single user mode, the 3B2 Computer will not power down if the power switch is depressed. The system can be powered down by executing the following:

```
# shutdown -i0 -g0 -y<CR>
```

Essential Command Exceptions

1. **The "fmtflop" verify fails when a certain sequence of events occurs.** When the following sequence of events takes place, the *verify* option of the *fmtflop* command will fail:
 1. The system board diagnostic phase 16 (floppy drive phase) is executed.
 2. The system is manually booted.
 3. The *fmtflop* command is executed (the *verify* will fail).

There are two possible work-arounds to the failure:

- A. Re-execute the *fmtflop* command
OR
B. After the system is booted, execute any command on the floppy (mount, labelit, etc.), then execute the *fmtflop* command.
2. **The "shutdown" command with the "-p" option is obsolete and should not be used.** The following command:
shutdown -p

is obsolete and should not be used.

3. **The " umountall " command with the " -k " option does not work correctly.** When the *umountall -k* command is entered, a response is never returned. You must depress the <break> key in order to regain use of the terminal. In order to simulate what the *umountall -k* command does, the following must be executed (as root):
 1. Execute a **ps -eaf** to get a list of all active processes
 2. Execute a **kill -9 pid** for all pids listed in the above *ps*
 3. Execute a **mount** to determine all mounted filesystems
 4. Execute a **umount filesystem** for all filesystems except root
4. **The " mkfs " command will accept any value for file system size.** One of the arguments to the *mkfs* command is the number of physical blocks that the file system will occupy. The *mkfs* command has no way of determining the size of the media. Therefore it will accept any value for the number of blocks, and it reports no error if the number of blocks is larger than the media. It is recommended that you use the Simple Administration facilities for making filesystems by entering the following:

```
# sysadm makefsys<CR>
```

5. **The " mkfs " command may create a filesystem with a bad free list.** When a filesystem is made with the filesystem size equal to two hundred 512-byte blocks, and the requested gap and interleave is 1 and f218 respectively, the filesystem will be created with a bad free list. An *fsck* of the filesystem will correct the freelist.

It is recommended that the Simple Administration interface always be used when making filesystems:

```
# sysadm makefsys<CR>
```

This command uses default values which will not cause the above problem.

6. **The " mount " command does not protect against over-mounts.** A partition which is already mounted can be re-mounted over another partition. This can corrupt data on the originally mounted filesystem. If the Simple Administration interface is used:

```
# sysadm mountfsys<CR>
```

the problem cannot occur.

Core UNIX System Exceptions

1. **System logins sys, nuucp, and rje will not work unless certain optional utilities are installed.** System logins are special logins used by very knowledgeable users to do special tasks. A part of setting up the 3B2 Computer is to assign passwords to these logins. In order to log in as **sys**, the Source Utilities must have been installed. The system login **nuucp**, requires the Basic Networking Utilities to be installed. To log in as **rje**, the 3BNET Utilities must have been installed.
2. **UNIX System V Release 1.0 floppies must be "labeled" before they will be recognized by Simple Administration commands of Release 2.0.** A floppy which was written on Release 1.0 must be *labeled* before the Release 2.0 Simple Administration commands will recognize it. Each Release 1.0 floppy should be inserted in the floppy drive, and the following command executed:

```
# labelit /dev/diskette fsname volume<CR>
```

The *fsname* is any name you choose to *name* the floppy, and the *volume* name is the volume name, or number, of the floppy. This function is automatically provided in the Release 2.0 Simple Administration command (makefsys).

3. **UNIX System V Release 1.0 floppies must be filesystem checked before being used on Release 2.0.** File systems which were made on Release 1.0 must be checked via the *fsck* command before being used on Release 2.0. The following command should be executed on all Release 2.0 floppies:

```
# sysadm checkfsys<CR>
```

The following will be displayed on the terminal:

```
/dev/diskette
File System: name Volume: volume
**Phase 1 - Check Blocks and Sizes
**Phase 2 - Check Pathnames
**Phase 3 - Check Connectivity
**Phase 4 - Check Reference Counts
**Phase 5 - Check Free List
FILE SYSTEM STATE SET TO OKAY
#files # blocks ##### free
*** FILE SYSTEM WAS MODIFIED ***
```

System Administration Utilities Exceptions

1. The "fuser" usage example in the documentation is incorrect.

The manual page for the *fuser* command is incorrect. It should read:

/etc/fuser -[ku] [file1 . . .]

The description of the arguments and all other related documentation is correct.

2. Crash does not work correctly when the buffer command is given "inode" as the format. When the crash *buffer* command is given *inode* as the format with which to print the buffer, it prints the buffer, then core dumps. The buffer is printed correctly.

3. **The "whodo" command may fail, unless the "ps" command has been executed.** Every time the system is initialized, the *ps* command must be executed before the *whodo* command is executed. If it has not been executed, the following error message will be generated:

```
whodo: open error of /etc/ps_data: No such file or directory
```

If the *ps* command is executed first, all subsequent *whodo* requests will execute properly until the system is reinitialized.

4. **The "ff" command when used with the "-u" option does not work.** When the *ff -u* command is executed, the following message is returned:

```
Memory fault: Core dump
```

and no output is returned from the command.

DOCUMENTATION

This Software Information Bulletin should be inserted in the *3B2 Computer Owner/Operator Manual*.

The system administration-type commands provided as part of the Essential Utilities and the commands provided by the System Administration Utilities are described in the *3B2 Computer System Administration Utilities Guide*. Also described in this guide are various system administration procedures (tasks).

RELEASE FORMAT

Storage Structure

The commands delivered as the System Administration Utilities are installed in the **/etc** directory.

System Requirements

The minimum equipment configuration required for the use of the System Administration Utilities is 0.5 megabytes of random access memory and a 10-megabyte hard disk.

To install the System Administration Utilities software, there must be 763 free blocks of storage in the **root** (/) file system and 47 free blocks of storage in the **/usr** file system. Adequate storage space is checked automatically as part of the installation process. The installation process installs the utilities only if adequate storage space is available.

The System Administration Utilities for the 3B2 Computer are distributed on one floppy disk. Most of the utilities are object code files. The **mvdir** command is a shell script. Other system administration-type shell scripts provided by the Essential Utilities are prototypes that you can change to suit your operating environment. Refer to the *3B2 Computer System Administration Utilities Guide* for further information on prototype files.

Files Delivered

The System Administration Utilities are delivered on a single floppy disk. The directory structure and files are as follows.

DIRECTORY	FILES			
/etc	checkall	ff	ldsysdump	sysdef
	chroot	fsdb	link	unlink
	crash	fsdb1b	mvdir	volcopy
	dcopy	fuser	ncheck	whodo
	dfsck	grpck	pwck	

UTILITIES INSTALL PROCEDURE

Use the standard software utilities install procedure described in the *3B2 Computer Owner/Operator Manual* for the installation of the System Administration Utilities.

UTILITIES REMOVE PROCEDURE

Use the standard software utilities remove procedure described in the *3B2 Computer Owner/Operator Manual* for the removal of the System Administration Utilities.

AT&T 3B2/300 COMPUTER
USER REFERENCE
MANUAL

Update Issue 1
December 10, 1984

This update inventory should be placed behind the *3B2 Computer User Reference Manual* title page. This inventory identifies the pages that have been added, deleted, or changed by the *3B2 Computer User Reference Manual Update (305-407)*, dated December 10, 1984.

Revised Page(s)	Date
tput manual pages	11/84

Added Page(s)	Date
ar manual pages	11/84
edtbl manual pages	11/84

Deleted Page(s)	Date
logdir manual pages	10/84
nkill manual pages	10/84

INTRODUCTION

This manual describes the features of the UNIX system. It provides neither a general overview of the UNIX system nor details of the implementation of the system.

This manual contains a single section named "Utilities" that is divided into interfiled subclasses:

1. Essential Utilities
2. Editing Utilities
3. Directory and File Management Utilities
4. Help Utilities
5. Terminal Information Utilities
6. User Environment Utilities

If you are a domestic U. S. customer, you have also received pages labeled Security Administration Utilities that should be interfiled in this document.

Section 1 (Utilities) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for binary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the **shell**. Some UNIX systems may have a directory called **/usr/lbin**, containing local commands.

The numbers following the command are intended for easy cross reference. A command with a **(1)** usually means that the command is contained in this manual. A command with a **(1C)** usually means that the command is a communications utility. A command with a **(1G)** usually means that the command is a graphics utility. A command with a **(1M)**, **(7)**, or **(8)** following it means that the command is in the appropriate section of the *AT&T 3B2 Computer System Administration Utilities Guide*. A command with a **(2)**, **(3)**, **(4)**, or **(5)** following it means that the command is in the appropriate section of the *AT&T 3B2 Computer System Programmer Reference Manual*.

Section 1 consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries are alphabetized, with the exception of the introductory entry that begins Section 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Utilities*):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets **[]** around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses **...** are used to show that the previous argument prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus **-**, plus **+**, or an equal sign **=** is often taken to be some sort of flag argument, even if it appears in a position where a file name could

appear. Therefore, it is unwise to have files whose names begin with `-`, `+`, or `=`.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index derived from that table precede Section 1. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. The *Permuted Index* is used by searching the middle column for a key word or phrase. The right column will then contain the name of the manual page that contains the command. The left column contains additional useful information about the command.

HOW TO GET STARTED

This discussion provides the basic information you need to get started on the UNIX system: how to log in and log out, how to communicate through your terminal, and how to run a program. (See the *UNIX System V User Guide* for a more complete introduction to the system.)

Logging in. You must dial up the UNIX operating system from an appropriate terminal. The UNIX system supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained (together with the telephone number(s) of your UNIX system) from the administrator of your system. Common terminal speeds are 10, 15, 30, and 120 characters per second (110, 150, 300, and 1200 baud); occasionally, speeds of 240, 480, and 960 characters per second (2400, 4800, and 9600 baud) are also available. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one "preferred" speed; if you dial in from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the `login:` message at the wrong speed). Keep hitting the "break" or "attention" key until the `login:` message appears. Hard-wired terminals usually are set to the correct speed.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system types `login:` and you then type your user name followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

It is important that you type your login name in lowercase if possible; if you type uppercase letters, the UNIX system will assume that your terminal cannot generate lowercase letters and that you mean all subsequent uppercase input to be treated as lowercase. When you have logged in successfully, the shell will type a `$` to you. (The shell is described below under *How to run a program*.)

For more information, consult `login(1)`, which discusses the login sequence in more detail, and `stty(1)`, which tells you how to describe the characteristics of your terminal to the system. The command (`profile(4)`) in The *UNIX System V Programmer Reference Manual* explains how to accomplish this last task automatically every time you log in).

Logging out. There are two ways to log out:

1. You can simply hang up the phone.
2. You can log out by typing an end-of-file indication (ASCII EOT character, usually typed as "control-d") to the shell. The shell will terminate and the `login:` message will appear again.

How to communicate through your terminal. When you type to the UNIX system, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in *Logging in*.

UNIX system terminal input/output is full-duplex. It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system silently throws away *all* the saved characters.

On an input line from a terminal, the character @ cancels all the characters typed before it on that line. The character # erases the last character typed. Successive uses of # will erase characters back to, but not beyond, the beginning of the line; @ and # can be typed as themselves by preceding them with \ (thus, to erase a \, you need two #s). These default erase and kill characters can be changed; see *stty(1)*.

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a DC1 (control-q) or a second DC3 (or any other character, for that matter) is typed. The DC1 and DC3 characters are not passed to any other program when used in this manner.

The ASCII DEL (a.k.a. "rubout") character is not passed to programs, but instead generates an *interrupt signal*, just like the "break", "interrupt", or "attention" signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you do not want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed(1)*, for example, catches interrupts and stops what *it* is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the ASCII FS character. It not only causes a running program to terminate, but also, if possible, generates a file with the "core image" of the terminated process. *Quit* is useful for debugging.

Besides adapting to the speed of the terminal, the UNIX system tries to be intelligent as to whether you have a terminal with the "new-line" function, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all *input* "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "line-feed" pair is echoed to the terminal. If you get into the wrong mode, the *stty(1)* command will rescue you.

Tab characters are used freely in UNIX system source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty(1)* command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs(1)* command will set tab stops on your terminal, if that is possible.

How to run a program. When you have successfully logged into the UNIX system, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and type a \$ at you to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh(1)*.

The current directory. The UNIX system has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is, by default, assumed to be in that directory. Because you are the owner of this directory, you have full permissions

to read, write, alter, or destroy its contents. Permissions to access and/or modify other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current directory use *cd(1)*.

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with */*, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next subdirectory (followed by a */*), until finally the file name is reached (e.g., */usr/ae/filex* refers to file *filex* in directory *ae*, while *ae* is itself a subdirectory of *usr*; *usr* springs directly from the root directory). See *intro(2)* for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed */*). Without important exception, a path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp(1)*, *mv*, and *rm(1)*, which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls(1)*. Use *mkdir(1)* for making directories and *rmdir(1)* for destroying them.

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above.

Writing a program. To enter the text of a source program into a UNIX system file, use *ed(1)*. After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named *a.out* (if that output is precious, use *mv(1)* to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see *ld(1)*).

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the \$ prompt.

Your programs can receive arguments from the command line just as system programs do; see *exec(2)*.

Text processing. Almost all text is entered through the editor *ed(1)*. The commands most often used to write text on a terminal are *cat(1)* or *pr(1)*. The *cat(1)* command simply dumps ASCII text on the terminal, with no processing at all. The *pr(1)* command paginates the text, supplies headings, and has a facility for multi-column output.

Surprises. Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, *write(1)* is used; *mail(1)* will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first \$.

TABLE OF CONTENTS

1. Utilities

intro	introduction to commands and application programs
at	execute commands at a later time
awk	pattern scanning and processing language
banner	make posters
basename	deliver portions of path names
bc	arbitrary-precision arithmetic language
bdf	big diff
bfs	big file scanner
cal	print calendar
calendar	reminder service
cat	concatenate and print files
cd	change working directory
chmod	change mode
chown	change owner or group
cmp	compare two files
comm	select or reject lines common to two sorted files
cp	copy, link or move files
cpio	copy file archives in and out
crontab	user crontab file
csplit	context split
cut	cut out selected fields of each line of a file
date	print and set the date
dc	desk calculator
diff	differential file comparator
diff3	3-way differential file comparison
dircomp	directory comparison
echo	echo arguments
ed	text editor
edit	text editor (variant of ex for casual users)
egrep	search a file for a pattern
env	set environment for command execution
ex	text editor
expr	evaluate arguments as an expression
factor	factor a number
fgrep	search a file for a pattern
file	determine file type
find	find files
getopt	parse command options
glossary	definitions of common UNIX system terms and symbols
grep	search a file for a pattern
help	UNIX System
helpadm	make changes to the
join	relational database operator
kill	terminate a process
line	read one line
locate	identify a
logdir	get login directory
login	sign on
logname	get login name
ls	list contents of directory
machid	provide truth value about your processor type
mail	send mail to users or read mail
mailx	interactive message processing system
mesg	permit or deny messages

Table of Contents

mkdir	make a directory
newform	change the format of a text file
news	print news items
nice	run a command at low priority
nkill	terminate a process
nl	line numbering filter
nohup	run a command immune to hangups and quits
od	octal dump
pack	compress and expand files
passwd	change login password
paste	merge same lines of several files or subsequent lines of one file
pg	file perusal filter for CRTs
pr	print files
ps	report process status
pwd	working directory name
rm	remove files or directories
sdiff	side-by-side difference program
sed	stream editor
setup	initialize system for first user
sh	shell, the standard/restricted command programming language
shl	shell layer manager
sleep	suspend execution for an interval
sort	sort and/or merge files
split	split a file into pieces
starter	information about the
stty	set the options for a terminal
sum	print checksum and block count of a file
sysadm	menu interface to do system administration
tabs	set tabs on a terminal
tail	deliver the last part of a file
tee	pipe fitting
test	condition evaluation command
time	time a command
touch	update access and modification times of a file
tput	query terminfo database
tr	translate characters
true	provide truth values
tty	get the name of the terminal
umask	set file-creation mode mask
uname	print name of current UNIX system
uniq	report repeated lines in a file
units	conversion program
usage	retrieve a command description and usage examples
vi	screen-oriented (visual) display editor based on ex
wait	await completion of process
wall	write to all users
wc	word count
who	who is on the system
write	write to another user
xargs	construct argument list(s) and execute command

PERMUTED INDEX

comparison. diff3:	3-way differential file	diff3(1)
of a file. touch:	update access and modification times	touch(1)
menu interface to do system	administration. sysadm:	sysadm(1)
sort: sort	and/or merge files.	sort(1)
introduction to commands and	application programs. intro:	intro(1)
maintainer for portable/	ar: archive and library	ar(1)
language. bc:	arbitrary-precision arithmetic	bc(1)
for portable archives. ar:	archive and library maintainer	ar(1)
maintainer for portable	archives. /archive and library	ar(1)
cpio: copy file	archives in and out.	cpio(1)
command. xargs: construct	argument list(s) and execute	xargs(1)
expr: evaluate	arguments as an expression.	expr(1)
echo: echo	arguments.	echo(1)
bc: arbitrary-precision	arithmetic language.	bc(1)
expr: evaluate arguments	as an expression.	expr(1)
a later time.	at, batch: execute commands at	at(1)
wait:	await completion of process.	wait(1)
processing language.	awk: pattern scanning and	awk(1)
(visual) display editor	banner: make posters.	banner(1)
portions of path names.	based on ex. /screen-oriented	vi(1)
later time. at,	basename, dirname: deliver	basename(1)
arithmetic language.	batch: execute commands at a	at(1)
sum: print checksum and	bc: arbitrary-precision	bc(1)
dc: desk	bdf: big diff.	bdf(1)
cal: print	bfs: big file scanner.	bfs(1)
text editor (variant of ex for	block count of a file.	sum(1)
files.	cal: print calendar.	cal(1)
tr: translate	calculator.	dc(1)
file. sum: print	calendar.	cal(1)
chown,	calendar: reminder service.	calendar(1)
group.	casual users. edit:	edit(1)
common to two sorted files.	cat: concatenate and print	cat(1)
nice: run a	cd: change working directory.	cd(1)
examples. usage: retrieve a	characters.	tr(1)
env: set environment for	checksum and block count of a	sum(1)
quits. nohup: run a	chgrp: change owner or group.	chown(1)
getopt: parse	chmod: change mode.	chmod(1)
/shell, the standard/restricted	chown, chgrp: change owner or	chown(1)
test: condition evaluation	cmp: compare two files.	cmp(1)
time: time a	comm: select or reject lines	comm(1)
argument list(s) and execute	command at low priority.	nice(1)
intro: introduction to	command description and usage	usage(1)
at, batch: execute	command execution.	env(1)
comm: select or reject lines	command immune to hangups and	nohup(1)
glossary: definitions of	command options.	getopt(1)
diff: differential file	command programming language.	sh(1)
cmp:	command.	test(1)
diff3: 3-way differential file	command.	time(1)
dircmp: directory	command. xargs: construct	xargs(1)
wait: await	commands and application/	intro(1)
pack, pcat, unpack:	commands at a later time.	at(1)
cat:	common to two sorted files.	comm(1)
test:	common UNIX system terms and/	glossary(1)
	comparator.	diff(1)
	compare two files.	cmp(1)
	comparison.	diff3(1)
	comparison.	diremp(1)
	completion of process.	wait(1)
	compress and expand files.	pack(1)
	concatenate and print files.	cat(1)
	condition evaluation command.	test(1)

execute command. xargs:	construct argument list(s) and	xargs(1)
ls: list	contents of directory.	ls(1)
csplit:	context split.	csplit(1)
units:	conversion program.	units(1)
cpio:	copy file archives in and out.	cpio(1)
cp, ln, mv:	copy, link or move files.	cp(1)
sum: print checksum and block	count of a file.	sum(1)
wc: word	count.	wc(1)
files.	cp, ln, mv: copy, link or move	cp(1)
and out.	cpio: copy file archives in	cpio(1)
crontab: user	crontab file.	crontab(1)
pg: file perusal filter for	crontab: user crontab file.	crontab(1)
uname: print name of	CRTs.	pg(1)
of each line of a file.	csplit: context split.	csplit(1)
each line of a file. cut:	current UNIX system.	uname(1)
join: relational	cut: cut out selected fields	cut(1)
tput: query terminfo	cut out selected fields of	cut(1)
date: print and set the	database operator.	join(1)
date.	database.	tput(1)
date: print and set the date.	date.	date(1)
dc: desk calculator.	date: print and set the date.	date(1)
definitions of common UNIX	dc: desk calculator.	dc(1)
names. basename, dirname:	deliver portions of path	basename(1)
file. tail:	deliver the last part of a	tail(1)
mesg: permit or	deny messages.	mesg(1)
usage: retrieve a command	description and usage/	usage(1)
dc:	desk calculator.	dc(1)
file:	determine file type.	file(1)
bdiff: big	diff.	bdiff(1)
comparator.	diff: differential file	diff(1)
comparison.	diff3: 3-way differential file	diff3(1)
sdiff: side-by-side	difference program.	sdiff(1)
diff:	differential file comparator.	diff(1)
diff3: 3-way	differential file comparison.	diff3(1)
rm, rmdir: remove files or	diremp: directory comparison.	diremp(1)
cd: change working	directories.	rm(1)
directory.	cd: change working	cd(1)
dircmp:	directory comparison.	dircmp(1)
logdir: get login	directory.	logdir(1)
ls: list contents of	directory.	ls(1)
mkdir: make a	directory.	mkdir(1)
pwd: working	directory name.	pwd(1)
path names. basename,	dirname: deliver portions of	basename(1)
vi: screen-oriented (visual)	display editor based on ex.	vi(1)
od: octal	dump.	od(1)
echo:	echo arguments.	echo(1)
ex for casual users).	echo: echo arguments.	echo(1)
/ (visual) display	ed, red: text editor.	ed(1)
ed, red: text	edit: text editor (variant of	edit(1)
ex: text	editor based on ex.	vi(1)
sed: stream	editor.	ed(1)
casual users). edit: text	editor.	ex(1)
pattern.	editor.	sed(1)
command execution.	editor (variant of ex for	edit(1)
execution. env: set	egrep: search a file for a	egrep(1)
expression. expr:	env: set environment for	env(1)
test: condition	environment for command	env(1)
edit: text editor (variant of	evaluate arguments as an	expr(1)
display editor based on	evaluation command.	test(1)
construct argument list(s) and	ex for casual users).	edit(1)
	ex: text editor.	ex(1)
	ex. /screen-oriented (visual)	vi(1)
	execute command. xargs:	xargs(1)

time. at, batch:	execute commands at a later	at(1)
set environment for command sleep: suspend	execution. env:	env(1)
pcat, unpack: compress and expression.	execution for an interval.	sleep(1)
expr: evaluate arguments as an factor:	expand files. pack,	pack(1)
true, pattern.	expr: evaluate arguments as an expression.	expr(1)
cpio: copy	factor a number.	expr(1)
diff: differential	factor: factor a number.	factor(1)
diff3: 3-way differential	false: provide truth values.	true(1)
crontab: user crontab	fgrep: search a file for a	fgrep(1)
fields of each line of a	file archives in and out.	cpio(1)
	file comparator.	diff(1)
	file comparison.	diff3(1)
	file.	crontab(1)
	file. cut: cut out selected	cut(1)
	file: determine file type.	file(1)
	file for a pattern.	egrep(1)
	file for a pattern.	fgrep(1)
	file for a pattern.	grep(1)
	file into pieces.	split(1)
change the format of a text or subsequent lines of one	file. newform:	newform(1)
	file. /lines of several files	paste(1)
	file perusal filter for CRTs.	pg(1)
	file scanner.	bfs(1)
	file. sum: print	sum(1)
	file. tail:	tail(1)
	file. touch: update access	touch(1)
	file type.	file(1)
	file. uniq:	uniq(1)
	file-creation mode mask.	umask(1)
	files.	cat(1)
	files.	cmp(1)
	files. comm: select or reject	comm(1)
	files.	cp(1)
	files.	find(1)
	files or directories.	rm(1)
	files or subsequent lines of/	paste(1)
	files. pack, pcat,	pack(1)
	files.	pr(1)
	files.	sort(1)
sort: sort and/or merge	filter for CRTs.	pg(1)
	filter.	nl(1)
	find files.	find(1)
	find: find files.	find(1)
	fitting.	tee(1)
newform: change the	format of a text file.	newform(1)
	logdir: get login directory.	logdir(1)
	logname: get login name.	logname(1)
	tty: get the name of the terminal.	tty(1)
	getopt: parse command options.	getopt(1)
	glossary: definitions of	glossary(1)
	grep: search a file for a	grep(1)
	group.	chown(1)
	hangups and quits.	nohup(1)
	help: UNIX System.	help(1)
	helpadm: make changes to the.	helpadm(1)
locate:	identify a.	locate(1)
nohup: run a command	immune to hangups and quits.	nohup(1)
	user. setup: initialize system for first	setup(1)
	system. mailx: interactive message processing	mailx(1)
administration. sysadm: menu	interface to do system	sysadm(1)
suspend execution for an commands and application/	interval. sleep:	sleep(1)
application programs. intro:	intro: introduction to	intro(1)
	introduction to commands and	intro(1)

news: print news	items.	news(1)
operator.	join: relational database	join(1)
scanning and processing	kill: terminate a process.	kill(1)
arbitrary-precision arithmetic	language. awk: pattern	awk(1)
command programming	language. bc:	bc(1)
shl: shell	language. /standard/restricted	sh(1)
portable/ ar: archive and	layer manager.	sh(1)
line: read one	library maintainer for	ar(1)
nl:	line.	line(1)
out selected fields of each	line numbering filter.	nl(1)
files. comm: select or reject	line of a file. cut: cut	cut(1)
uniq: report repeated	line: read one line.	line(1)
of several files or subsequent	lines common to two sorted	comm(1)
subsequent/ paste: merge same	lines in a file.	uniq(1)
cp, ln, mv: copy,	lines of one file. /same lines	paste(1)
ls:	lines of several files or	paste(1)
xargs: construct argument	link or move files.	cp(1)
files. cp,	list contents of directory.	ls(1)
logdir: get	list(s) and execute command.	xargs(1)
logname: get	ln, mv: copy, link or move	cp(1)
passwd: change	locate: identify a.	locate(1)
login directory.	logdir: get login directory.	logdir(1)
nice: run a command at	login directory.	logdir(1)
directory.	login name.	logname(1)
send mail to users or read	login password.	passwd(1)
users or read mail.	login: sign on.	login(1)
mail, rmail: send	logname: get login name.	logname(1)
processing system.	low priority.	nice(1)
ar: archive and library	ls: list contents of	ls(1)
mkdir:	mail, mail, rmail:	mail(1)
helpadm:	mail, rmail: send mail to	mail(1)
banner:	mail to users or read mail.	mail(1)
shl: shell layer	mailx: interactive message	mailx(1)
umask: set file-creation mode	maintainer for portable/	ar(1)
administration. sysadm:	make a directory.	mkdir(1)
sort: sort and/or	make changes to the.	helpadm(1)
files or subsequent/ paste:	make posters.	banner(1)
mailx: interactive	manager.	shl(1)
mesg: permit or deny	mask.	umask(1)
chmod: change	menu interface to do system	sysadm(1)
umask: set file-creation	merge files.	sort(1)
touch: update access and	merge same lines of several	paste(1)
cp, ln, mv: copy, link or	mesg: permit or deny messages.	mesg(1)
cp, ln,	message processing system.	mailx(1)
a text file.	messages.	mesg(1)
news: print	mkdir: make a directory.	mkdir(1)
priority.	mode.	chmod(1)
hangups and quits.	mode mask.	umask(1)
nl: line	modification times of a file.	touch(1)
od:	move files.	\ep(1)
join: relational database	mv: copy, link or move files.	cp(1)
stty: set the	newform: change the format of	newform(1)
getopt: parse command	news items.	news(1)
	news: print news items.	news(1)
	nice: run a command at low	nice(1)
	nkill: terminate a process.	nkill(1)
	nl: line numbering filter.	nl(1)
	nohup: run a command immune to	nohup(1)
	od: octal dump.	od(1)
	operator.	join(1)
	options for a terminal.	stty(1)
	options.	getopt(1)

chown, chgrp: change and expand files.	owner or group.	chown(1)
getopt:	pack, pcat, unpack: compress	pack(1)
passwd: change login several files or subsequent/	parse command options.	getopt(1)
dirname: deliver portions of	passwd: change login password.	passwd(1)
egrep: search a file for a	password.	passwd(1)
fgrep: search a file for a	paste: merge same lines of	paste(1)
grep: search a file for a	path names. basename,	basename(1)
processing language. awk:	pattern.	egrep(1)
expand files. pack,	pattern.	fgrep(1)
provide truth value about/	pattern.	grep(1)
msgs:	pattern scanning and	awk(1)
pg:	pcat, unpack: compress and	pack(1)
CRTs.	pdp11, u3b, u3b2, u3b5, vax:	machid(1)
split: split a file into	permit or deny messages.	mesg(1)
tee:	perusal filter for CRTs.	pg(1)
and library maintainer for	pg: file perusal filter for	pg(1)
basename, dirname: deliver	pieces.	split(1)
banner: make	pipe fitting.	tee(1)
date:	portable archives. /archive	ar(1)
cal:	portions of path names.	basename(1)
of a file. sum:	posters.	banner(1)
cat: concatenate and	pr: print files.	pr(1)
pr:	print and set the date.	date(1)
system, uname:	print calendar.	cal(1)
news:	print checksum and block count	sum(1)
nice: run a command at low	print files.	cat(1)
kill: terminate a	print files.	pr(1)
nkill: terminate a	print name of current UNIX	uname(1)
ps: report	print news items.	news(1)
wait: await completion of	priority.	nice(1)
awk: pattern scanning and	process.	kill(1)
mailx: interactive message	process.	nkill(1)
provide truth value about your	process status.	ps(1)
standard/restricted command	process.	wait(1)
pdp11, u3b, u3b2, u3b5, vax:	processing language.	awk(1)
true, false:	processing system.	mailx(1)
tput:	processor type. /u3b5, vax:	machid(1)
command immune to hangups and	programming language. /the	sh(1)
rmail: send mail to users or	provide truth value about your/	machid(1)
line:	provide truth values.	true(1)
ed:	ps: report process status.	ps(1)
sorted files. comm: select or	pwd: working directory name.	pwd(1)
join:	query terminfo database.	tput(1)
calendar:	quits. nohup: run a	nohup(1)
rm, rmdir:	read mail. mail,	mail(1)
uniq: report	read one line.	line(1)
ps:	red: text editor.	ed(1)
file. uniq:	reject lines common to two	comm(1)
and usage examples. usage:	relational database operator.	join(1)
directories:	reminder service.	calendar(1)
read mail. mail,	remove files or directories.	rm(1)
directories. rm,	repeated lines in a file.	uniq(1)
standard/restricted/ sh,	report process status.	ps(1)
nice:	report repeated lines in a	uniq(1)
hangups and quits. nohup:	retrieve a command description	usage(1)
bfs: big file	rm, rmdir: remove files or	rm(1)
language. awk: pattern	rmail: send mail to users or	mail(1)
display editor based on/ vi:	rmdir: remove files or	rm(1)
	rsh: shell, the	sh(1)
	run a command at low priority.	nice(1)
	run a command immune to	nohup(1)
	scanner.	bfs(1)
	scanning and processing	awk(1)
	screen-oriented (visual)	vi(1)

program.	sdiff: side-by-side difference	sdiff(1)
egrep:	search a file for a pattern.	egrep(1)
fgrep:	search a file for a pattern.	fgrep(1)
grep:	search a file for a pattern.	grep(1)
	sed: stream editor.	sed(1)
to two sorted files. comm:	select or reject lines common	comm(1)
of a file. cut: cut out selected fields of each line	cut(1)	
mail. mail, rmail:	send mail to users or read	mail(1)
first user.	setup: initialize system for	setup(1)
standard/restricted command/	sh, rsh: shell, the	sh(1)
shl:	shell layer manager.	shl(1)
command programming/	sh, rsh: shell, the standard/restricted	sh(1)
shl: shell layer manager.	shl(1)	
program. sdiff:	side-by-side difference	sdiff(1)
login:	sign on.	login(1)
an interval.	sleep: suspend execution for	sleep(1)
sort:	sort and/or merge files.	sort(1)
sort: sort and/or merge files.	sort(1)	
sorted files. comm: select	comm(1)	
or reject lines common to two split:	split a file into pieces.	split(1)
split: context pieces.	split.	split(1)
sh, rsh: shell, the	split: split a file into	split(1)
the.	standard/restricted command/	sh(1)
starter: information about	starter(1)	
ps: report process status.	ps(1)
sed: stream editor.	sed(1)	
terminal.	stty: set the options for a	stty(1)
/same lines of several files or count of a file.	subsequent lines of one file.	paste(1)
interval. sleep:	sum: print checksum and block	sum(1)
common UNIX system terms and system administration.	suspend execution for an	sleep(1)
tabs: set tabs on a file.	symbols. /definitions of	glossary(1)
stty: set the options for a terminal.	sysadm: menu interface to do	sysadm(1)
tabs: set tabs on a terminal.	tabs on a terminal.	tabs(1)
tty: get the name of the terminal.	tabs: set tabs on a terminal.	tabs(1)
kill:	tail: deliver the last part of	tail(1)
nkill:	tee: pipe fitting.	tee(1)
tput: query terminfo database.	terminal.	stty(1)
/of common UNIX system command.	terminal.	tabs(1)
ed, red:	terminal.	tty(1)
ex:	terminate a process.	kill(1)
casual users). edit: change the format of a file.	terminate a process.	nkill(1)
execute commands at a later time:	terms and symbols.	tput(1)
update access and modification times of a file.	test: condition evaluation	glossary(1)
values.	text editor.	test(1)
u3b, u3b2, u3b5, vax: provide true, false: provide terminal.	text editor (variant of ex for	ed(1)
file: determine file type.	text editor.	ex(1)
value about your processor truth value about your/ pdp11,	text file. newform:	edit(1)
truth value about your/ pdp11, value about your/ pdp11, u3b, u3b2, u3b5, vax: provide truth	time a command.	newform(1)
u3b, u3b2, u3b5, vax: provide truth values.	time, at, batch:	time(1)
truth values.	touch: time a command.	at(1)
tty: get the name of the terminal.	times of a file. touch:	time(1)
type.	touch: update access and	touch(1)
type. /vax: provide truth	tput: query terminfo database.	touch(1)
type.	tr: translate characters.	tput(1)
type.	translate characters.	tr(1)
type.	true, false: provide truth	tr(1)
type.	truth value about your/ pdp11,	true(1)
type.	truth values.	machid(1)
type.	tty: get the name of the	true(1)
type.	type.	tty(1)
type.	type. /vax: provide truth	file(1)
type.	type.	machid(1)
type.	type.	machid(1)
type.	type.	machid(1)

about your/ pdp11, u3b, u3b2,	u3b5, vax: provide truth value	machid(1)
mask.	umask: set file-creation mode	umask(1)
UNIX system.	uname: print name of current	uname(1)
a file.	uniq: report repeated lines in	uniq(1)
	units: conversion program.	units(1)
files. pack, pcat,	unpack: compress and expand	pack(1)
times of a file. touch:	update access and modification	touch(1)
a command description and	usage examples. /retrieve	usage(1)
description and usage/	usage: retrieve a command	usage(1)
crontab:	user crontab file.	crontab(1)
initialize system for first	user. setup:	setup(1)
write: write to another	user.	write(1)
(variant of ex for casual	users). edit: text editor	edit(1)
mail, rmail: send mail to	users or read mail.	mail(1)
wall: write to all	users.	wall(1)
/u3b2, u3b5, vax: provide truth	value about your processor/	machid(1)
true, false: provide truth	values.	true(1)
users). edit: text editor	(variant of ex for casual	edit(1)
your/ pdp11, u3b, u3b2, u3b5,	vax: provide truth value about	machid(1)
display editor based on ex.	vi: screen-oriented (visual)	vi(1)
on ex. vi: screen-oriented	(visual) display editor based	vi(1)
process.	wait: await completion of	wait(1)
	wall: write to all users.	wall(1)
	wc: word count.	wc(1)
who:	who is on the system.	who(1)
	who: who is on the system.	who(1)
cd: change	working directory.	cd(1)
	pwd: working directory name.	pwd(1)
	wall: write to all users.	wall(1)
	write: write to another user.	write(1)
	write: write to another user.	write(1)
list(s) and execute command.	xargs: construct argument	xargs(1)

NAME

ar — archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] afile [name] ...

DESCRIPTION

The *Ar* command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by *ar* consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When *ar* creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in *ar(4)*. The archive symbol table [described in *ar(4)*] is used by the link editor [*ld(1)*] to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table is only created and maintained by *ar* when there is at least one object file in the archive. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. Whenever the *ar(1)* command is used to create or update the contents of such an archive, the symbol table is rebuilt. The *s* option described below will force the symbol table to be rebuilt.

Key is an optional *-*, followed by one character from the set **drqtpmx**, optionally concatenated with one or more of **vuaibcls**. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.
- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.
- v** Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.
- c** Suppress the message that is produced by default when *afie* is created.

- I** Place temporary files in the local current working directory, rather than in the directory specified by the environment variable TMPDIR or in the default directory /tmp.
- s** Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

FILES

/tmp/ar* temporaries

SEE ALSO

ld(1), *lorder*(1), *strip*(1).
tmpnam(3S), *a.out*(4), *ar*(4) in the *3B2 Computer System Programmer Reference Manual*.

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

at, batch – execute commands at a later time

SYNOPSIS

at *time* [*date*] [+ *increment*]
at -r *job...*
at -l [*job...*]

batch

DESCRIPTION

At and *batch* read commands from standard input to be executed at a later time. The *sh(1)* utility provides different way of specifying standard input. *At* allows you to specify when the commands should be executed, while jobs queued with *batch* will execute when system load level permits. *At* may be used with the following options:

- r Removes jobs previously scheduled with *at*.
- l Reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use *at* if their name appears in the file */usr/lib/cron/at.allow*. If that file does not exist, the file */usr/lib/cron/at.deny* is checked to determine if the user should be denied access to *at*. If neither file exists, only root is allowed to submit a job. If *at.deny* is empty, global usage is permitted. The allow/deny files consist of one user name per line.

The *time* may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning *hour:minute*. A suffix **am** or **pm** may be appended; otherwise a 24-hour clock time is understood. The suffix **zulu** may be used to indicate GMT. The special names **noon**, **midnight**, **now**, and **next** are also recognized.

An optional *date* may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special “days”, **today** and **tomorrow** are recognized. If no *date* is given, **today** is assumed if the given hour is greater than the current hour and **tomorrow** is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional *increment* is simply a number suffixed by one of the following: **minutes**, **hours**, **days**, **weeks**, **months**, or **years**. (The singular form is also accepted.)

Thus legitimate commands include:

```
at 0815am Jan 24
at 8:15am Jan 24
at now + 1 day
at 5 pm Friday
```

At and *batch* write the job number and schedule time to standard error.

Batch submits a batch job. It is almost equivalent to “at now”, but not quite. For one, it goes into a different queue. For another, “at now” will respond with the error message **too late**.

At -r removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at -l*. You can only remove your own jobs unless you are the super-user.

EXAMPLES

The *at* and *batch* commands read from standard input the commands to be executed at a later time. *Sh(1)* provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch  
sort filename > outfile  
<control-D> (hold down 'control' and depress 'D')
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!  
sort filename 2>&1 > outfile | mail loginid  
!
```

To have a job reschedule itself, invoke *at* from within the shell procedure, by including code similar to the following within the shell file:

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

kill(1), *mail(1)*, *nice(1)*, *ps(1)*, *sh(1)*, *sort(1)*.
cron(1M) in the *3B2 Computer System Administration Utilities Guide*.

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk — pattern scanning and processing language

SYNOPSIS

awk [*-Fc*] [*prog*] [*parameters*] [*files*]

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as *-f file*. The *prog* string should be enclosed in single quotes ('') to protect it from the shell.

Parameters, in the form *x=... y=...* etc., may be passed to *awk*.

Files are read in order; if there are no files, the standard input is read. The file name — means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted \$1, \$2, ...; \$0 refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /, %, and concatenation (indicated by a blank). The C operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted *x[i]*) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted ("").

The *print* statement prints its arguments on the standard output (or on a file if *>expr* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format [see *printf(3S)* in the 3B2 Computer]

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr(s, m, n)* returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf(fmt, expr, expr, ...)* formats the expressions according to the *printf(3S)* format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, & &, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep* (see *grep(1)*). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (for *contains*) or !~ (for *does not contain*). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

```
BEGIN { FS = c }
```

or by using the *-Fc* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default %.6g).

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
{ print }
```

command line: awk -f program n=5 input

SEE ALSO

grep(1), sed(1).
malloc(3X), printf(3S) in the *3B2 Computer System Programmer Reference Manual*.
lex(1) in the *3B2 Computer System Extended Software Generation System Utilities*.

BUGS

Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.

NAME

banner — make posters

SYNOPSIS

banner strings

DESCRIPTION

Banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1).

NAME

basename, dirname — deliver portions of path names

SYNOPSIS

basename *string* [*suffix*]
dirname *string*

DESCRIPTION

Basename deletes any prefix ending in / and the *suffix* (if present in *string*) from *string*, and prints the result on the standard output. It is normally used inside substitution marks (` `) within shell procedures.

Dirname delivers all but the last level of the path name in *string*.

EXAMPLES

The following example, invoked with the argument /usr/src/cmd/cat.c, compiles the named file and moves the output to a file named cat in the current directory:

```
cc $1  
mv a.out `basename $1 '\.c'
```

The following example will set the shell variable NAME to /usr/src/cmd:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

SEE ALSO

sh(1).

BUGS

The *basename* of / is null and is considered an error.

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

bc [**-c**] [**-l**] [file ...]

DESCRIPTION

Bc is an interactive processor for a language that resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The **bc**(1) utility is actually a preprocessor for **dc**(1), which it invokes automatically unless the **-c** option is present. In this case the **dc** input is sent to the standard output instead. The options are as follows:

-c Compile only. The output is send to the standard output.

-l Argument stands for the name of an arbitrary precision math library.

The syntax for *bc* programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

are enclosed in /* and */.

Names

simple variables: L

array elements: L [E]

The words "ibase", "obase", and "scale"

Other operands

arbitrarily long numbers with optional sign and decimal point.

(E)

sqrt (E)

length (E) number of significant decimal digits

scale (E) number of digits right of decimal point

L (E , ... , E)

Operators

+ - * / % ^ (% is remainder; ^ is power)

+ + -- (prefix and postfix; apply to names)

= = < = > = != < >

= = + = - = * = / = % = ^

Statements

E

{ S ; ... ; S }

if (E) S

while (E) S

for (E ; E ; E) S

null statement

break

quit

Function definitions

```
define L ( L ,..., L ) {
    auto L, ... , L
    S; ... S
    return ( E )
}
```

Functions in **-l** math library

s(x) sine

c(x) cosine

e(x) exponential

l(x) log

a(x)	arctangent
j(n,x)	Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc(1)*. Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

EXAMPLE

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; i==1; i++){
        a = a*x
        b = b*i
        c = a/b
        if(c == 0) return(s)
        s = s+c
    }
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

FILES

/usr/lib/lib.b	mathematical library
/usr/bin/dc	desk calculator proper

SEE ALSO

dc(1).

BUGS

No &&, || yet.

For statement must have all three expressions (E's).

Quit is interpreted when read, not when executed.

NAME

bdiff — big diff

SYNOPSIS

bdiff file1 file2 [n] [-s]

DESCRIPTION

Bdiff is used in a manner analogous to *diff(1)* to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*.

The parameters to *bdiff* are:

file1 (*file2*)

The name of a file to be used. If *file1* (*file2*) is **-**, the standard input is read.

n The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail.

-s Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible exclamations by *diff*.

Bdiff ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd?????

SEE ALSO

diff(1), *help(1)*.

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

bfs — big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

The *Bfs* command is (almost) like *ed(1)* except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 512 characters, including new-line, per line (255 for 16-bit machines). *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit(1)* can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the **w** command. The optional **-** suppresses printing of sizes. Input is prompted with * if P and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another P and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters a through z may be used, and all 26 marks are remembered.

The e, g, v, k, p, q, w, =, ! and null commands operate as described under *ed*. Commands such as ---, +++, ++=, -12, and +4p are accepted. Note that 1,10p and 1,10 will both print the first ten lines. The f command only prints the name of the file being scanned; there is no *remembered* file name. The w command is independent of output diversion, truncation, or crunching (see the xo, xt and xc commands, below). The following additional commands are available:

xf *file*

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the xf. The xf commands may be nested to a depth of 10.

xn List the marks currently in use (marks are set by the k command).

xo [*file*]

Further output from the p and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

:label

This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(. . .)xb/regular expression/label

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and \$.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, . is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

xb/^/ label

is an unconditional jump.

The xb command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt number

Output from the p and null commands is truncated to at most *number* characters. The initial number is 255.

xv[digit][spaces][value]

The variable name is the specified *digit* following the xv. The commands xv5100 or xv5 100 both assign the value 100 to the variable 5. The command Xv61,100p assigns the value 1,100p to the variable 6. To reference a variable, put a % in front of the variable name. For example, using the above assignments for variables 5 and 6:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

g/%5/p

would globally search for the characters 100 and print each line containing a match. To escape the special meaning of %, a \ must precede it.

g/" *%\[cds\]/p

could be used to match and list lines containing printf of characters, decimal integers, or strings.

Another feature of the xv command is that the first line of output from a UNIX system command can be stored into a variable. The only requirement is that the first character of *value* be an !. For example:

```
.w junk
xv5!cat junk
!rm junk
!echo "%5"
xv6!expr %6 + 1
```

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a ****.

xv7\!date

stores the value **!date** into variable **7**.

xbz label

xbn label

These two commands will test the last saved *return code* from the execution of a UNIX system command (**!command**) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

```
xv55  
:1  
/size/  
xv5!expr %5 - 1  
!if 0%5 != 0 exit 2  
xbn 1  
xv45  
:1  
/size/  
xv4!expr %4 - 1  
!if 0%4 = 0 exit 2  
xbz 1
```

xc [switch]

If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** it is not. Without an argument, **xc** reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), ed(1).

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

NAME

cal - print calendar

SYNOPSIS

cal [[month] year]

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar — reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Aug. 24," "august 24," "8/24," etc., are recognized, but not "24 August" or "24/8". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in the login directory and sends them any positive results by **mail(1)**. Normally this is done daily by facilities in the UNIX operating system.

FILES

/usr/lib/calprog	to figure out today's and tomorrow's dates
/etc/passwd	
/tmp/cal*	

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service.
Calendar's extended idea of "tomorrow" does not account for holidays.

NAME

cat — concatenate and print files

SYNOPSIS

cat [**-u**] [**-s**] [**-v** [**-t**] [**-e**]] *file* ...

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

cat *file*

prints the file, and:

cat *file1* *file2* >*file3*

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument **-** is encountered, *cat* reads from the standard input file.

The following options apply to *cat*.

- u** The output is not buffered. (The default is buffered output.)
- s** *Cat* is silent about non-existent files.
- v** Causes non-printing characters (with the exception of tabs, new-lines and form-feeds) to be printed visibly. Control characters are printed ^*X* (control-*x*); the DEL character (octal 0177) is printed ^?. Non-ASCII characters (with the high bit set) are printed as M-*x*, where *x* is the character specified by the seven low order bits.

When used with the **-v** option, the following options may be used.

- t** Causes tabs to be printed as ^T's.
- e** Causes a \$ character to be printed at the end of each line (prior to the new-line).

The **-t** and **-e** options are ignored if the **-v** option is not specified.

WARNING

Command formats such as

cat *file1* *file2* >*file1*

will cause the original data in *file1* to be lost; therefore, take care when using shell special characters.

SEE ALSO

cp(1), **pg(1)**, **pr(1)**.

NAME

cd — change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

If *directory* is not specified, the value of shell parameter \$HOME is used as the new working directory. If *directory* specifies a complete path starting with /, ., .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the \$CDPATH shell variable. \$CDPATH has the same syntax as, and similar semantics to, the \$PATH shell variable. Cd must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and is internal to the shell.

SEE ALSO

pwd(1), *sh(1)*.

chdir(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

chmod — change mode

SYNOPSIS

chmod mode files

DESCRIPTION

The permissions of the named *files* are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see <i>chmod(2)</i>
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for **ugo**, the default if *who* is omitted.

Op can be **+** to add *permission* to the file's mode, **-** to take away *permission*, or **=** to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **s** (set owner or group ID) and **t** (save text, or sticky); **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with **=** to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g** and **t** only works with **u**.

Only the owner of a file (or the super-user) may change its mode. Only the super-user may set the sticky bit. In order to set the group ID, the group of the file must correspond to your current group ID.

EXAMPLES

The first example denies write permission to others, the second makes a file executable:

chmod o-w file

chmod +x file

SEE ALSO

ls(1).

chmod(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

chown, chgrp — change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

FILES

/etc/passwd

/etc/group

SEE ALSO

chmod(1).

chown(2), group(4), passwd(4) in the 3B2 Computer System Programmer Reference Manual.

NAME

cmp — compare two files

SYNOPSIS

cmp [**-l**] [**-s**] file1 file2

DESCRIPTION

The two files are compared. (If *file1* is **-**, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

- l** Print the byte number (decimal) and the differing bytes (octal) for each difference.
- s** Print nothing for differing files; return codes only.

SEE ALSO

comm(1), *diff*(1).

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

NAME

comm — select or reject lines common to two sorted files

SYNOPSIS

comm [- [123]] file1 file2

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort(1)*), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name **-** means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus **comm -12** prints only the lines common to the two files; **comm -23** prints only lines in the first file but not in the second; **comm -123** is a prints nothing.

SEE ALSO

cmp(1), **diff(1)**, **sort(1)**, **uniq(1)**.

NAME

cp, ln, mv — copy, link or move files

SYNOPSIS

```
cp file1 [ file2 ...] target  
ln [ -f ] file1 [ file2 ...] target  
mv [ -f ] file1 [ file2 ...] target
```

DESCRIPTION

file1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same (take care when using *sh(1)* metacharacters). If *target* is a directory, then one or more files are copied (linked, moved) to that directory. If *target* is a file, its contents are destroyed.

If *mv* or *ln* determines that the mode of *target* forbids writing, it will print the mode (see *chmod(2)*), ask for a response, and read the standard input for one line; if the line begins with *y*, the *mv* or *ln* occurs, if permissible; if not, the command exits. No questions are asked and the *mv* or *ln* is done when the *-f* option is used or if the standard input is not a terminal.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent; *file1* is renamed *target*. If *file1* is a file and *target* is a link to another file with links, the other links remain and *target* becomes a new file.

When using *cp*, if *target* is not a file, a new file is created which has the same mode as *file1* except that the sticky bit is not set unless you are super-user; the owner and group of *target* are those of the user. If *target* is a file, copying a file into *target* does not change its mode, owner, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.

SEE ALSO

chmod(1), *cpio(1)*, *rm(1)*.

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case any linking relationship with other files is lost.

Ln will not link across file systems.

NAME

cpio — copy file archives in and out

SYNOPSIS

```
cpio -o [ acBv ]
cpio -i [ BedmrtuvsB ] [ patterns ]
cpio -p [ adlmrv ] directory
```

DESCRIPTION

Cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary.

Cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous **cpio -o**. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of *sh(1)*. In *patterns*, meta-characters ?, *, and [...] match the slash / character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio -o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes **cpio** to retain the owner and group of the files of the previous **cpio -o**.

Cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

- a** Reset access times of input files after they have been copied.
- B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from */dev/rmt/??*).
- d** Directories are to be created as needed.
- c** Write *header* information in ASCII character form for portability.
- r** Interactively *rename* files. If the user types a null line, the file is skipped.
- t** Print a *table of contents* of the input. No files are created.
- u** Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
- v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an **ls -l** command (see *ls(1)*).
- l** Whenever possible, link files rather than copying them. Usable only with the **-p** option.
- m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- f** Copy in all files except those in *patterns*.
- s** Swap bytes. Use only with the **-i** option.
- S** Swap halfwords. Use only with the **-i** option.
- b** Swap both bytes and halfwords. Use only with the **-i** option.

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/mt/0m  
cd olddir  
find . -depth -print | cpio -pdl newdir
```

The trivial case “`find . -depth -print | cpio -oB >/dev/rmt/0m`” can be handled more efficiently by:

```
find . -cpio /dev/rmt/0m
```

SEE ALSO

`ar(1)`, `find(1)`, `ls(1)`.

`cpio(4)` in the *3B2 Computer System Programmer Reference Manual*.

BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files.

NAME

crontab — user crontab file

SYNOPSIS

crontab [file]
crontab -r
crontab -l

DESCRIPTION

Crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The *-r* option removes a user's crontab from the crontab directory. *Crontab -l* will list the crontab file for the invoking user.

Users are permitted to use *crontab* if their names appear in the file */usr/lib/cron/cron.allow*. If that file does not exist, the file */usr/lib/cron/cron.deny* is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If either file is at *cron.deny*, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

minute (0–59),
hour (0–23),
day of the month (1–31),
month of the year (1–12),
day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your \$HOME directory with an arg0 of sh. Users who desire to have their .profile executed must explicitly do so in the crontab file. Cron supplies a default environment for every shell, defining HOME, LOGNAME, SHELL(=/bin/sh), and PATH(=:/bin:/usr/bin:/usr/lbin).

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

SEE ALSO

sh(1).

cron(1M) in the *3B2 Computer System Administration Utilities Guide*.

NAME

csplit — context split

SYNOPSIS

csplit [-s] [-k] [-f prefix] file arg1 [... argn]

DESCRIPTION

Csplit reads *file* and separates it into $n+1$ sections, defined by the arguments *arg1* ... *argn*. By default the sections are placed in *xx00* ... *xxn* (*n* may not be greater than 99). These sections get the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.
- ⋮
- n+1*: From the line referenced by *argn* to the end of *file*.

If the *file* argument is a **-** then standard input is used.

The options to *csplit* are:

- s** *Csplit* normally prints the character counts for each file created. If the **-s** option is present, *csplit* suppresses the printing of all character counts.
- k** *Csplit* normally removes created files if an error occurs. If the **-k** option is present, *csplit* leaves previously created files intact.
- f prefix** If the **-f** option is used, the created files are named *prefix00* ... *prefixn*. The default is **xx00** ... *xxn*.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

- /rexpl* A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *rexpl*. The current line becomes the line containing *rexpl*. This argument may be followed by an optional + or - some number of lines (e.g., */Page/-5*).
- %rexpl%* This argument is the same as */rexpl*, except that no file is created for the section.
- Inno* A file is to be created from the current line up to (but not including) *Inno*. The current line becomes *Inno*.
- {num}* Repeat argument. This argument may follow any of the above arguments. If it follows a *rexpl* type argument, that argument is applied *num* more times. If it follows *Inno*, the file will be split every *Inno* lines (*num* times) from that point.

Enclose all *rexpl* type arguments that contain blanks or other characters meaningful to the shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00** ... **cobol03**. After editing the “split” files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%' '/^}/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

SEE ALSO

ed(1), **sh(1)**.
regexp(5) in the *3B2 Computer System Programmer Reference Manual*.
regexp(5).

DIAGNOSTICS

Self-explanatory except for:

arg — out of range

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

cut — cut out selected fields of each line of a file

SYNOPSIS

```
cut -clist [file1 file2 ...]
cut -flist [-dchar] [-s] [file1 file2 ...]
```

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (*-c* option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (*-f* option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

- list*** A comma-separated list of integer field numbers (in increasing order), with optional **-** to indicate ranges [e.g., **1,4,7; 1-3,8; -5,10** (short for **1-5,10**); or **3-** (short for third through last field)].
- clist*** The *list* following **-c** (no space) specifies character positions (e.g., **-c1-72** would pass the first 72 characters of each line).
- flist*** The *list* following **-f** is a list of fields assumed to be separated in the file by a delimiter character (see **-d**); e.g., **-f1,7** copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless **-s** is specified.
- dchar*** The character following **-d** is the field delimiter (**-f** option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
- s*** Suppresses lines with no delimiter characters in case of **-f** option. Unless specified, lines with no delimiters will be passed through untouched.

Either the **-c** or **-f** option must be specified.

Use *grep(1)* to make horizontal “cuts” (by context) through a file, or *paste(1)* to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

cut -d: -f1,5 /etc/passwd	mapping of user IDs to names
name=`who am i cut -f1 -d" "`	to set name to current login name.

DIAGNOSTICS

- line too long*** A line can have no more than 1023 characters or fields.
- bad list for c/f option*** Missing **-c** or **-f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.
- no fields*** The *list* is empty.

SEE ALSO

grep(1), *paste(1)*.

NAME

date — print and set the date

SYNOPSIS

date [mmddhhmm[yy]] | +format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year — 01 to 12
d	day of month — 01 to 31
y	last 2 digits of year — 00 to 99
D	date as mm/dd/yy
H	hour — 00 to 23
M	minute — 00 to 59
S	second — 00 to 59
T	time as HH:MM:SS
j	day of year — 001 to 366
w	day of week — Sunday = 0
a	abbreviated weekday — Sun to Sat
h	abbreviated month — Jan to Dec
r	time in AM/PM notation

EXAMPLE

date '+DATE: %m/%d/%y%TIME: %H:%M:%S'

would have generated as output:

DATE: 08/01/76

TIME: 14:45:05

DIAGNOSTICS

<i>No permission</i>	if you are not the super-user and you try to change the date;
<i>bad conversion</i>	if the date set is syntactically incorrect;
<i>bad format character</i>	if the field descriptor is not recognizable.

FILES

/dev/kmem

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dc — desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *Bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0–9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ - / * % ^

The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s_x The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the s is capitalized, *x* is treated as a stack and the value is pushed on it.

l_x The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the l is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. P interprets the top of the stack as an ASCII string, removes it, and prints it.

f All values on the stack are printed.

q exits the program. If executing a string, the recursion level is popped by two. If q is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of *dc* commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ASCII string onto the top of the stack.

<*x* >*x* =*x*

The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

! interprets the rest of the line as a UNIX system command.

- c All values on the stack are popped.
- i The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.
- o The top value on the stack is popped and used as the number radix for further output.
- O pushes the output base on the top of the stack.
- k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
- z The stack level is pushed onto the stack.
- Z replaces the number on the top of the stack with its length.
- ? A line of input is taken from the input source (usually the terminal) and executed.
- :: are used by bc for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
Osa1
lyx
```

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented

where *x* is an octal number.

stack empty

for not enough elements on the stack to do what was asked.

Out of space

when the free list is exhausted (too many digits).

Out of headers

for too many numbers being kept around.

Out of pushdown

for too many items on the stack.

Nesting Depth

for too many levels of nested execution.

NAME

diff — differential file comparator

SYNOPSIS

diff [**-efbh**] *file1* *file2*

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is **-**, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging *a* for *d* and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by **<**, then all the lines that are affected in the second file flagged by **>**.

The **-b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **-e** option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **-f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file (*\$1*) and a chain of version-to-version *ed* scripts (*\$2,\$3,...*) made by *diff* need be on hand. A “latest version” appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **-h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **-e** and **-f** are unavailable with **-h**.

FILES

```
/tmp/d?????
/usr/lib/difff for -h
```

SEE ALSO

cmp(1), *comm*(1), *ed*(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

Editing scripts produced under the **-e** or **-f** option are naive about creating lines consisting of a single period (.)

WARNINGS

Missing newline at end of file X

indicates that the last line of file *X* did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

NAME

diff3 — 3-way differential file comparison

SYNOPSIS

diff3 [**-ex3**] *file1* *file2* *file3*

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

=====	all three files differ
=====1	<i>file1</i> is different
=====2	<i>file2</i> is different
=====3	<i>file3</i> is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f : *n1* a Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.

f : *n1* , *n2* c Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a c indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the **-e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ===== and =====3. Option **-x** (-3) produces a script to incorporate only changes flagged ===== (=====3). The following command will apply the resulting script to *file1*.

(cat script; echo '1,\$p') | ed - *file1*

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat **-e**.
Files longer than 64K bytes will not work.

NAME

diremp — directory comparison

SYNOPSIS

diremp [-d] [-s] [-wn] dir1 dir2

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

- d** Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff(1)*.
- s** Suppress messages about identical files.
- wn** Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), *diff(1)*.

NAME

echo — echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\\	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <i>n</i> , which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

NAME

ed, *red* — text editor

SYNOPSIS

```
ed [ - ] [ -p string ] [ file ]
red [ - ] [ -p string ] [ file ]
```

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional *-* suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !shell command. The *-p* option allows the user to specify a prompt string. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via !shell command. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec(4)* formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in *stty -tabs* or *stty tab3* mode (see *stty(1)*, the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.
- 1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
 - a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]); see 1.4 below).

- b. ^ (caret or circumflex), which is special at the *beginning* of an *entire RE* (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).
 - c. \$ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).
 - d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)
- 1.3 A period (.) is a one-character RE that matches any character except new-line.
- 1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [la-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *REs* from one-character *REs*:

- 2.1 A one-character RE is a RE that matches whatever the one-character RE matches.
- 2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.
- 2.3 A one-character RE followed by \{m\}, \{m,\}, or \{m,n\} is a RE that matches a *range* of occurrences of the one-character RE. The values of m and n must be non-negative integers less than 256; \{m\} matches *exactly* m occurrences; \{m,\} matches *at least* m occurrences; \{m,n\} matches *any number* of occurrences *between* m and n inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.
- 2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.
- 2.5 A RE enclosed between the character sequences \(` and \)` is a RE that matches whatever the unadorned RE matches.
- 2.6 The expression \n matches the same string of characters as was matched by an expression enclosed between \(` and \)` *earlier* in the same RE. Here n is a digit; the sub-expression specified is that beginning with the n-th occurrence of \(` counting from the left. For example, the expression ^\(.*\)`\`1\$ matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both).

- 3.1 A circumflex (^) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

- 3.2 A currency symbol (\$) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction \wedge *entire RE\$* constrains the entire RE to match the entire line. The null RE (e.g., //) is equivalent to the last RE encountered. See also the last paragraph before FILES below.

To understand addressing in ed it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character \$ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x*' addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before FILES below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before FILES below.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean .-5.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,\$, while a semicolon (;) stands for the pair .,\$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,), They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by *l*, *n*, or *p* in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a

<text>

The *append* command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.)c

<text>

The *change* command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(.,.)d

The *delete* command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e *file*

The *edit* command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh(1)*) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E *file*

The *Edit* command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f *file*

If *file* is given, the *file-name* command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,\$)g/RE/command list

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \; *a*, *i*, and *c* commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,\$)G/RE/

In the interactive Global command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an & causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *help* command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *Help* command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i

<text>

The *insert* command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j

The *join* command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)kx

The *mark* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x*' then addresses this line; . is unchanged.

(.,.l)

The *list* command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.ma)

The *move* command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(.,.n)

The *number* command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(.,.p)

The *print* command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially off.

q

The *quit* command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(\$.)r *file*

The *read* command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh(1)*) command whose output is to be read. For example, "\$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.s/*RE/replacement/* or(.,.s/*RE/replacement/g* or(.,.s/*RE/replacement/n* n = 1-512

The *substitute* command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number *n* appears after the command, only the *n* th occurrence of the matched string on each addressed line

is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters \n, where n is a digit, are replaced by the text matched by the n-th regular subexpression of the specified RE enclosed between \C and \V. When nested parenthesized subexpressions are present, n is determined by counting occurrences of \C starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a g or v command list.

(.,.)ta

This command acts just like the m command, except that a *copy* of the addressed lines is placed after address a (which may be 0); . is left at the last line of the copy.

u

The undo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent a, c, d, g, i, j, m, r, s, t, v, G, or V command.

(1,\$)v/RE/command list

This command is the same as the global command g except that the *command list* is executed with . initially set to every line that does *not* match the RE.

(1,\$)V/RE/

This command is the same as the interactive global command G except that the lines that are marked during the first step are those that do *not* match the RE.

(1,\$)w file

The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your umask setting (see sh(1)) dictates otherwise. The currently-remembered file name is *not* changed unless file is the very first file name mentioned since ed was invoked. If no file name is given, the currently-remembered file name, if any, is used (see e and f commands); . is unchanged. If the command is successful, the number of characters written is typed. If file is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

(\$)=

The line number of the addressed line is typed; . is unchanged by this command.

!shell command

The remainder of the line after the ! is sent to the UNIX system shell (*sh(1)*) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to .+1p; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a ? and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, *ed* discards ASCII NUL characters and all characters after the last new-line. Files (e.g., *a.out*) that contain characters not in the ASCII set (bit 8 on) cannot be edited by *ed*.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

/tmp/e# temporary; # is the process number.
ed.hup work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.
?file for an inaccessible file.
(use the help and Help commands for detailed explanations).

If changes have been made in the buffer since the last w command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the e or q commands. It prints ? and allows one to continue editing. A second e or q command at this point will take effect. The - command-line option inhibits this feature.

SEE ALSO

grep(1), *sed(1)*, *sh(1)*, *stty(1)*.
fspec(4), *regexp(5)* in the *3B2 Computer System Programmer Reference Manual*.

BUGS

A ! command cannot be subject to a g or a v command.
The ! command and the ! escape from the e, r, and w commands cannot be used if the the editor is invoked from a restricted shell (see sh(1)).
The sequence \n in a RE does not match a new-line character.
The l command mishandles DEL.
Characters are masked to 7 bits on input.
If the editor input is coming from a command file (i.e., ed file < ed-cmd-file), the editor will exit at the first failure of a command that is in the command file.

NAME

edit — text editor (variant of ex for casual users)

SYNOPSIS

edit [-r] name ...

DESCRIPTION

Edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor.

-r Recover file after an editor or system crash.

The following brief introduction should help you get started with *edit*. If you are using a CRT terminal you may want to learn about the display editor *vi*.

To edit the contents of an existing file you begin with the command “*edit name*” to the shell. *Edit* makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run *edit* on it; you will cause an error diagnostic, but do not worry.

Edit prompts for commands with the character ‘:’, which you should see after starting the editor. If you are editing an existing file, then you will have some lines in *edit*’s buffer (its name for the copy of the file you are editing). Most commands to *edit* use its “current line” if you do not tell them which line to use. Thus if you say *print* (which can be abbreviated *p*) and hit carriage return (as you should after all *edit* commands) this current line will be printed. If you *delete* (*d*) the current line, *edit* will print the new current line. When you start editing, *edit* makes the last line of the file the current line. If you *delete* this last line, then the new last line becomes the current one. In general, after a *delete*, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the *append* (*a*) command can be used. After you give this command (typing a carriage return after the word *append*) *edit* will read lines from your terminal until you give a line consisting of just a “.”, placing these lines after the current line. The last line you type then becomes the current line. The command *insert* (*i*) is like *append* but places the lines you give before, rather than after, the current line.

Edit numbers the lines in the buffer, with the first line having number 1. If you give the command “1” then *edit* will type this first line. If you then give the command *delete* *edit* will delete the first line, line 2 will become line 1, and *edit* will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the *substitute* (*s*) command. You say “*s/old/new/*” where *old* is replaced by the old characters you want to get rid of and *new* is the new characters you want to replace it with.

The command *file* (*f*) will tell you how many lines there are in the buffer you are editing and will say “[Modified]” if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a *write* (*w*) command. You can then leave the editor by issuing a *quit* (*q*) command. If you run *edit* on a file, but do not change it, it is not necessary (but does no harm) to *write* the file back. If you try to *quit* from *edit* after modifying the buffer without writing it out, you will be warned that there has been “No write since last change” and *edit* will await another command. If you wish not to *write* the buffer out then you can issue another *quit* command. The buffer is then irretrievably discarded, and you return to the shell.

By using the **delete** and **append** commands, and giving line numbers to see lines in the file you can make any changes you desire. You should learn at least a few more things, however, if you are to use *edit* more than a few times.

The **change** (c) command will change the current line to a sequence of lines you supply (as in **append** you give lines up to a line consisting of only a "."). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, i.e., "3,5change". You can print lines this way too. Thus "1,23p" prints the first 23 lines of the file.

The **undo** (u) command will reverse the effect of the last command you gave which changed the buffer. Thus if you give a **substitute** command which does not do what you want, you can say **undo** and the old contents of the line will be restored. You can also **undo** an **undo** command so that you can continue to change your mind. *Edit* will give you a warning message when commands you do affect more than one line of the buffer. If the amount of change seems unreasonable, you should consider doing an **undo** and looking to see what happened. If you decide that the change is ok, then you can **undo** again to get it back. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a number of lines hit 'D (control key and, while it is held down D key, then let up both) rather than carriage return. This will show you a half screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at the text around where you are by giving the command "z.". The current line will then be the last line printed; you can get back to the line where you were before the "z." command by saying "". The z command can also be given other following characters "z—" prints a screen of text (or 24 lines) ending where you are; "z+" prints the next screenful. If you want less than a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts works in general; thus you can delete 5 lines starting with the current line with the command "delete 5".

To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form /text/ to search forward for *text* or ?text? to search backward for *text*. If a search reaches the end of the file without finding the text it wraps, end around, and continues to search back to the line where you are. A useful feature here is a search of the form /text/ which searches for *text* at the beginning of a line. Similarly /text\$/ searches for *text* at the end of a line. You can leave off the trailing / or ? in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in "..,\$print" which prints the rest of the lines in the file. To get to the last line in the file you can refer to it by its symbolic name "\$". Thus the command "\$ delete" or "\$d" deletes the last line in the file, no matter which line was the current line before. Arithmetic with line references is also possible. Thus the line "\$-5" is the fifth before the last, and ".+20" is 20 lines after the present.

You can find out which line you are at by doing ".=". This is useful if you wish to move or copy a section of text within a file or between files. Find out the first and last line numbers you wish to copy or move (say 10 to 20). For a move you can then say "10,20delete a" which deletes these lines from the file and places them in a buffer named *a*. *Edit* has 26 such buffers named *a* through *z*. You can later get these lines back by doing "put *a*" to put the contents of buffer *a* after the current line. If you want to move or copy these lines between files you can give an **edit** (e) command after copying the lines,

following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing *delete* to *yank* above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move \$" for example. It is not necessary to use named buffers in this case (but you can if you wish).

SEE ALSO

ex(1), vi(1).

NAME

`edittbl` — edit `edt_data` file

SYNOPSIS

`edittbl -d|s [-g] [-i] [-l] [-r] [-t] [file]`

DESCRIPTION

Edittbl is a user level utility that permits changes to `edt_data`, the file in the root file system **FILLEDT**. `Edt_data` is read during the Equipped Device Table (EDT) completion to get the device and subdevice look-up tables. This utility permits independent selection of device or subdevice tables, generation of either base table, new entry installation for either table, entry removal for the device table, and entry listings for either or both tables.

Edittbl prints the option list if the command has no arguments. The arguments are:

- d** This option selects the device look-up table for the utility's operation(s).
- s** This option selects the subdevice look-up table for the utility's operation(s).
- g** This option will generate the base look-up table entries for the selected look-up table(s). For the device table, these base entries are SBD, and PORTS. For the subdevice table, they are NULL, FD5, HD10, and HD30.
- i** This option specifies that new entries are to be added to the selected table. The ID codes for table entries and the input are compared; only new codes are installed. The formats for entries are described below. An EOF or “.” end the data input.
- l** This option specifies that the selected table(s) are listed.
- r** This option specifies that entries are to be removed from the device look-up table. When removing subdevice look-up table entries from `init/dgn/edt_data` conjunction with removing a device entry, this utility will check the Equipped Device Table (EDT) to verify that no subdevices specified for removal are present. The ID codes of the table are compared to the input and entries are removed for matches. The format is identical to that for the **-i** option and is listed below. An EOF or “.” end the data input.
- t** This option suppresses the program headings and user prompts; warnings and errors are not affected. This option is primarily useful in installation and removal scripts.
- file** The user may specify a target path name for the utilities. If none is specified, `./edt_data` is the default.

INPUT FORMAT

Data for installation/removal are entered as hex format numbers or character strings, one line for each table entry. The data fields must be supplied in the sequence described.

Devices

ID code	This field is a number between 0x0 and 0xffff that a device uses to identify itself. ID codes are administered by AT&T Technologies.
name	This field is a character string (maximum of 9 characters) that holds the user-recognizable name for a device. Device names are administered by AT&T Technologies. This string is also the file name that DGMON loads to diagnose a device.

rq_size	This is a number between 0x0 and 0xff for the count of entries in a device's job request queue.
cq_size	This is a number between 0x0 and 0xff for the count of entries in a device's job completion queue.
boot device	This field determines whether a device may be used to boot programs. A "1" means that it is bootable; a "0" means that it is not.
word size	This field shows the word size of a device I/O bus. A "1" is used for devices with a 16 bit bus word; a "0" is used for devices with an 8 bit bus word.
brd size	This field specifies the I/O connector slots that a device requires. A "1" indicates that two slots are needed; while a "0" means that one is required.
smart board	This field determines whether a device is intelligent, i.e., requires downloaded code for normal operation or supports subdevices. A "1" indicates an intelligent device, while a "0" specifies a "dumb" device.
cons_cap	This field shows whether a device can support the system console terminal. A "1" is used for devices that can; a "0" for those that cannot.
cons_file	This field shows whether a device requires pump code to provide a system console interface. A "1" in this field means that the board cannot support the console interface without extra code. This field may have the "1" value only when the cons_cap field does also. A "0" in this field means that the device can support a system console terminal with PROM-based code when cons_cap has the value "1". This field must have a "0" value when cons_cap is "0".
Subdevices	
ID code	This is a number between 0x0 and 0xffff for the code that identifies a subdevice. Subdevice ID codes are administered by AT&T Technologies.
subdev name	This field is a string (maximum of 9 characters) for a subdevice name. Subdevice names are uppercase and are administered by AT&T Technologies.
dev name	This field is a string (maximum of 9 characters) for the device name to which a subdevice is associated. If a device table entry is to be removed, associated subdevice table entries may also be removed in a separate program call. The device name is necessary for an Equipped Device Table (EDT) check that will verify that a subdevice table entry is needed only for a device entry that is to be removed.

EXAMPLES

Generate and list the base entries for both the device and subdevice tables, saving the results in **./edit_data**.

```
edittbl -g -l -s -d
```

Install subdevice entries with new ID codes from the file **subdev.in** into the existing file **./edit_data**.

```
edittbl -i -s < subdev.in
```

List the device table entries found in an existing copy of the file that DGMON loads, the INIT file system **edt_data** file.

```
edittbl -l -d /init/dgn/edt_data
```


NAME

egrep — search a file for a pattern

SYNOPSIS

egrep [options] [expression] [files]

DESCRIPTION

The *egrep* command searches the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. The following *options* are recognized:

- v All lines but those matching are printed.
- Same as a simple *expression* argument, but useful when the *expression* begins with a -
- f *file* The regular *expression* is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters \$, *, [, ^,], (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Egrep accepts regular expressions as in *ed(1)*, except for \C and \V, with the addition of:

1. A regular expression followed by + matches one or more occurrences of the regular expression.
2. A regular expression followed by ? matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by | or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is |, then *?+, then concatenation, then | and new-line.

SEE ALSO

ed(1), *fgrep(1)*, *grep(1)*, *sed(1)*, *sh(1)*.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Lines are limited to 256 characters; longer lines are truncated.

Egrep does not recognize ranges, such as [a-z], in character classes.

NAME

env — set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name*=*value* are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1).

exec(2), **environ(5)**, **profile(4)** in the *3B2 Computer System Programmer Reference Manual*.

NAME

ex — text editor

SYNOPSIS

ex [-] [-v] [-t *tag*] [-r] [-R] [+*command*] *name* ...

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base (see *Terminal Information Utilities Guide*) and the type of the terminal you are using from the variable TERM in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its visual command (which can be abbreviated *vi*) and which is the central mode of editing when using *vi*(1).

Ex contains a number of new features for easily viewing the text of the file. The z command gives easy access to windows of text. Hitting ^D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen-oriented visual mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The undo (*u*) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you do not accidentally clobber with a write a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor recover command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the next (*n*) command to deal with each in turn. The next command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command & in *ex* which repeats the last substitute command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is

convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent *join* (j) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

- Suppress all interactive-user feedback. This is useful in processing editor scripts.
- v Invokes *vi*
- t *tagR* Edit the file containing the *tag* and position the editor at its definition.
- r *file* Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- R *Readonly* mode set, prevents accidentally overwriting the file.
- +*command* Begin editing by executing the specified editor search or positioning *command*.

The *name* argument indicates files to be edited.

Ex States

Command	Normal and initial state. Input prompted for by :. Your kill character cancels partial command.
Insert	Entered by a i and c. Arbitrary text may be entered. Insert is normally terminated by line having only . on it, or abnormally with an interrupt.
Visual	Entered by vi, terminates with Q or ^\.

Ex command names and abbreviations

abbrev	ab	next	n	unabbrev	una
append	a	number	nu	undo	u
args	ar			unmap	unm
change	c	preserve	pre	version	ve
copy	co	print	p	visual	vi
delete	d	put	pu	write	w
edit	e	quit	q	xit	x
file	f	read	re	yank	ya
global	g	recover	rec	window	z
insert	i	rewind	rew	escape	!
join	j	set	se	lshift	<
list	l	shell	sh	print next	CR
map		source	so	resubst	&
mark	ma	stop	st	rshift	>
move	m	substitute	s	scroll	^D

Ex Command Addresses

n	line n	/pat	next with pat
.	current	?pat	previous with pat
\$	last	x-n	n before x
+	next	x,y	x through y
-	previous	'x	marked with x
+n	n forward	"	previous context
%	1,\$		

Initializing options

EXINIT	place set's here in environment var.
\$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set x	enable option
set nox	disable option
set x=val	give value val
set	show changed options
set all	show all options
set x?	show value of option x

Most useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
list		print `I for tab, \$ at end
magic		. I * special in patterns
number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ^D
showmatch	sm	to) and } as typed
showmode	smd	show insert mode in vi
slowopen	slow	stop updates during insert
window		visual mode lines
wrapscan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

	beginning of line
\$	end of line
.	any character
\<	beginning of word
\>	end of word
[str]	any char in str
[^str]	... not in str
[x-y]	... between x and y
*	any number of preceding

AUTHOR

Vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/usr/lib/*/*	describes capabilities of terminals
\$HOME/.exrc	editor startup file
./exrc	editor startup file
/tmp/Exnnnnn	editor temporary
/tmp/Rxnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), *ed*(1), *edit*(1), *grep*(1), *sed*(1), *vi*(1).
curses(3X), *term*(4), *terminfo*(4) in the *3B2 Computer System Programmer Reference Manual*.
Terminal Information Utilities Guide.

BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line ‘-’ option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

expr — evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2's complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by ****. The list is in order of increasing precedence, with equal precedence operators grouped within {} symbols.

expr \| expr

returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr \& expr

returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, >, >=, <, <=, != } expr

returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, - } expr

addition or subtraction of integer-valued arguments.

expr { *, /, % } expr

multiplication, division, or remainder of the integer-valued arguments.

expr : expr

The matching operator : compares the first argument with the second argument which must be a regular expression. Regular expression syntax is the same as that of *ed(1)*, except that all patterns are "anchored" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\| pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1. **a=\`expr \$a + 1\`**

adds 1 to the shell variable **a**.

2. **# For \$a equal to either "/usr/abc/file" or just "file"**
expr \$a : '.*\(.*\)' \| \$a

returns the last segment of a path name (i.e., file). Watch out for / alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. # A better representation of example 2.
`expr //\$a : '.*\(.*)'`

The addition of the `//` characters eliminates any ambiguity about the division operator and simplifies the whole expression.

4. `expr $VAR : '.*'`
returns the number of characters in `$VAR`.

SEE ALSO

`ed(1)`, `sh(1)`.

DIAGNOSTICS

As a side effect of expression evaluation, `expr` returns the following exit values:

- 0 if the expression is neither null nor 0
- 1 if the expression is null or 0
- 2 for invalid expressions.

syntax error for operator/operand errors

non-numeric argument if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, `expr` cannot tell the difference between an operator and an operand except by the value. If `$a` is an `=`, the command:

`expr $a = '='`

looks like:

`expr = = =`

as the arguments are passed to `expr` (and they will all be taken as the `=` operator). The following works:

`expr X$a = X=`

NAME

factor — factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{56} (about 1.0×10^{14}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to \sqrt{n} and occurs when *n* is prime or the square of a prime.

DIAGNOSTICS

“Ouch” for input out of range or for garbage input.

NAME

fgrep — search a file for a pattern

SYNOPSIS

fgrep [options] [strings] [files]

DESCRIPTION

Commands of the *frep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

- v All lines but those matching are printed.
- x (Exact) only lines matched in their entirety are printed
- c Only a count of matching lines is printed.
- l Only the names of files with matching lines are listed (once), separated by new-lines.
- n Each line is preceded by its relative line number in the file.
- b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
- e *expression* Same as a simple *expression* argument, but useful when the *expression* begins with a -.
- f *file* The regular *strings* list is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters \$, *, [, ^,], (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

SEE ALSO

ed(1), *sed*(1), *sh*(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *fgrep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to 256 characters; longer lines are truncated.

NAME

file — determine file type

SYNOPSIS

file [**-c**] [**-f** *file*] [**-m** *mfile*] *arg* ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0.

- c** The **-c** option causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **-c**.
- f** If the **-f** option is given, the next argument is taken to be a file containing the names of the files to be examined.
- m** The **-m** option instructs *file* to use an alternate magic file.

File uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

NAME

find — find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where *+n* means more than *n*, *-n* means less than *n* and *n* means exactly *n*. Valid expressions are:

- name** *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for !, ?, and *).
- perm** *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod(1)*). If *onum* is prefixed by a minus sign, more flag bits [017777, see *stat(2)* in the 3B2 Computer] become significant and the flags are compared.
- type** *c* True if the type of the file is *c*, where *c* is b, c, d, p, or f for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file respectively.
- links** *n* True if the file has *n* links.
- user** *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the /etc/passwd file, it is taken as a user ID.
- group** *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the /etc/group file, it is taken as a group ID.
- size** *n[c]* True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a *c*, the size is in characters.
- atime** *n* True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.
- mtime** *n* True if the file has been modified in *n* days.
- ctime** *n* True if the file has been changed in *n* days.
- exec** *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semi-colon. A command argument {} is replaced by the current path name.
- ok** *cmd* Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing y.
- print** Always true; causes the current path name to be printed.
- cpio** *device* Always true; write the current file on *device* in *cpio(1)* format (5120-byte records).
- newer** *file* True if the current file has been modified more recently than the argument *file*.

-depth Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio(1)* to transfer files that are contained in directories without write permission.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

- 1) The negation of a primary (! is the unary *not* operator).
- 2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- 3) Alternation of primaries (-o is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \(-name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1).

fs(4), stat(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

getopt — parse command options

SYNOPSIS

```
set -- `getopt optstring $*`
```

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *Optstring* is a string of recognized option letters (see *getopt(3C)*); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option **--** is used to delimit the end of the options. If it is used explicitly, *getopt* will recognize it; otherwise, *getopt* will generate it; in either case, *getopt* will place it at the end of the options. The positional parameters (\$1 \$2 ...) of the shell are reset so that each option is preceded by a **-** and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
        -a | -b)      FLAG=$i; shift;;
        -o)          OARG=$2; shift 2;;
        --)          shift; break;;
    esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

sh(1).

getopt(3C) in the *3B2 Computer System Programmer Reference Manual*.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

NAME

glossary — definitions of common UNIX system terms and symbols

SYNOPSIS

[*help*] *glossary* [*term*]

DESCRIPTION

The UNIX System *help* Facility command *glossary* provides definitions of common technical terms and symbols.

Without an argument, *glossary* displays a menu screen listing the terms and symbols that are currently included in *glossary*. A user may choose one of the terms or may exit to the shell by typing q (for "quit"). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition may also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following is a table which list the symbols and their escape sequence.

SYMBOL	ESCAPE SEQUENCE
""	\\""
"	\\''
[]	\[\]\\]
"	\\"`
#	\#
&	\&
*	*
\	\\\\\\
	\\\\\\\

From any screen in the facility, a user may execute a command via the shell (*sh(1)*) by typing a ! and the command to be executed. The screen will be redrawn if the command that was entered was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the *help* facility scrolls the data that is presented to the user. If a user prefers to have the screen clear before printing the data (non-scrolling), a variable must be defined in the user's .profile file called SCROLL. The variable SCROLL must be set to no and exported for non-scrolling to occur. If the user later decides that scrolling is desired, the variable SCROLL must be set to yes or deleted form the user's .profile file.

Further information on the UNIX System *help* Facility can be found on the *help(1)*, *usage(1)*, *starter(1)*, and *locate(1)* manual pages.

SEE ALSO

help(1), *locate(1)*, *sh(1)*, *starter(1)*, *usage(1)*.
term(5) in the 3B2 Computer System Programmer Reference Manual.

WARNINGS

If the TERM variable is not set in the user's .profile file, then terminal type will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term(5)* in the 3B2 Computer System Programmer Reference Manual. The *help* facility assumes that tabs are set on the user's terminal.

NAME

grep — search a file for a pattern

SYNOPSIS

grep [options] expression [files]

DESCRIPTION

The *fgrep* command searches the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed(1)*; it uses a compact non-deterministic algorithm. The following *options* are recognized:

- v** All lines but those matching are printed.
- x** (Exact) only lines matched in their entirety are printed (*fgrep* only).
- c** Only a count of matching lines is printed.
- i** Ignore upper/lower case distinction during comparisons.
- l** Only the names of files with matching lines are listed (once), separated by new-lines.
- n** Each line is preceded by its relative line number in the file.
- b** Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
- s** The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters \$, *, [, ^, |, (,), and \ in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

SEE ALSO

ed(1), *egrep(1)*, *fgrep(1)*, *egrep(1)*, *grep(1)*, *sed(1)*, *sh(1)*.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Lines are limited to BUFSIZ characters; longer lines are truncated. (BUFSIZ is defined in */usr/include/stdio.h*)

If there is a line with embedded nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

help — UNIX System help facility

SYNOPSIS

```

help
[ help ] starter
[ help ] usage [ -d ] [ -e ] [ -o ] [ command_name ]
[ help ] locate [ keyword1 [ keyword2 ] ... ]
[ help ] glossary [ term ]
help arg1 [ arg2 ... ]

```

DESCRIPTION

The UNIX System *help* facility provides on-line assistance for UNIX system users.

Without arguments, *help* prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are:

COMMAND	DESCRIPTION
<i>starter</i>	information about the UNIX system for the beginning user
<i>usage</i>	UNIX system command usage information
<i>locate</i>	locate UNIX system commands using function-related keywords
<i>glossary</i>	definitions of UNIX system technical terms

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

With arguments, *help* directly invokes the named on-line assistance command, bypassing the initial *help* menu. The commands *starter*, *locate*, *usage*, and *glossary*, optionally preceded by the word *help*, may also be specified at shell level. When executing *glossary* from shell level some of the symbols listed in the glossary must be escaped to be understood by the facility. For a list of symbols refer to the *glossary(1)* manual page.

From any screen in the facility, a user may execute a command via the shell (*sh(1)*) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the *help* facility scrolls the data that is presented to the user. If a user prefers to have the screen clear before printing the data (non-scrolling), a variable must be defined in the user's .profile file called SCROLL. The variable SCROLL must be set to no and exported for non-scrolling to occur. IF the user later decides that scrolling is desired, the variable SCROLL must be set to yes or deleted from the user's .profile file.

If the first argument to *help* is different from the four mentioned above, *help* assumes information is being requested in the form of the previous help command (often referred to as the SCCS help command, and now obsolete). The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type1 Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., ge3 for message 3 from the *get* command).

type2 Does not contain numerics (as a command, such as get).

type3 Is all numeric (e.g., 212).

Further information on the *starter*, *locate*, *usage*, and *glossary* commands may be found on the *starter(1)*, *locate(1)*, *usage(1)*, and *glossary(1)* manual pages, respectively.

SEE ALSO

get(1), *glossary(1)*, *locate(1)*, *sh(1)*, *starter(1)*, *usage(1)*.

term(5) in the *3B2 Computer System Programmer Reference Manual*.

WARNINGS

If the TERM variable is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term(5)*. The help facility assumes that tabs are set on the user's terminal.

NAME

helpadm — make changes to the *help* database

SYNOPSIS

helpadm

DESCRIPTION

The UNIX System *help* Facility Administration command **helpadm** allows UNIX system administrators and command developers to define the content of *help* for their specific commands and to monitor use of the *help* facility. The **helpadm** command can only be executed by login root, login bin, or a login that is a member of group bin.

The **helpadm** command prints a menu of 3 types of *help* data which can be modified, and 2 choices relating to monitoring use of the *help* facility. The five choices are:

- modify *startup* data
- add, modify, or delete a *glossary* term
- add, modify, or delete command data (description, options, examples, and keywords)
- prevent monitoring use of the *help* facility (login root and login bin only)
- permit monitoring use of the *help* facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, then they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, **helpadm** will put the user into *ed* to make additions/modifications to database information. If the user wishes to be put into a different editor, then they should set the EDITOR variable in their environment to the desired editor, and then export EDITOR.

If the user chooses to monitor/prevent monitoring use of the *help* facility, no further interaction occurs between the user and the *help* administration utilities.

SEE ALSO

ed(1), *help*(1).

WARNINGS

When the UNIX System is delivered to a customer, */etc/profile* exports the LOGNAME variable. If */etc/profile* has been changed so that LOGNAME is not exported, then the options to monitor/prevent monitoring use of the *help* facility may not work properly.

NAME

intro — introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands delivered with the AT&T 3B2 Computer. Certain distinctions of purpose are made in the headings:

- (1) Essential Utilities.
- (2) Editing Utilities.
- (3) Directory and File Management Utilities.
- (4) Help Utilities.
- (5) Terminal Information Utilities.
- (6) User Environment Utilities.

COMMAND SYNTAX

Unless otherwise noted, commands described accept options and other arguments according to the following syntax:

name [*option(s)*] [*cmdarg(s)*]

where:

<i>name</i>	The name of an executable file.
<i>option</i>	<ul style="list-style-type: none"> — <i>noargletter(s)</i> or, — <i>argletter <> optarg</i> where <> is optional white space.
<i>noargletter</i>	A single letter representing an option without an argument.
<i>argletter</i>	A single letter representing an option requiring an argument.
<i>optarg</i>	Argument (character string) satisfying preceding <i>argletter</i> .
<i>cmdarg</i>	Path name (or other command argument) <i>not</i> beginning with — or, — by itself indicating the standard input.

SEE ALSO

getopt(1).

exit(2), *getopt(3C)*, *wait(2)* in the *3B2 Computer System Programmer Reference Manual*.

How to Get Started, at the front of this volume.

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait(2)* and *exit(2)*). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

join — relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is **-**, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see *sort(1)*].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

- an** In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- e s** Replace empty output fields by string *s*.
- jn m** Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file. Fields are numbered starting with 1.
- o list** Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number. The common field is not printed unless specifically requested.
- tc** Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant. The character *c* is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

```
join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group
```

SEE ALSO

awk(1), *comm(1)*, *sort(1)*, *uniq(1)*.

BUGS

With default field separation, the collating sequence is that of *sort -b*; with **-t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk(1)* are wildly incongruous.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

NAME

kill — terminate a process

SYNOPSIS

kill [-signo] PID ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by - is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill -9 ..." is a sure kill.

SEE ALSO

ps(1), *sh*(1).

kill(2), *signal*(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

line — read one line

SYNOPSIS

line

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1).

read(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

locate - identify a UNIX system command using keywords

SYNOPSIS

[help] locate
[help] locate [keyword1 [keyword2] ...]

DESCRIPTION

The *locate* command is part of the UNIX system *help* Facility, and provides on-line assistance with identifying UNIX system commands.

Without arguments, the initial *locate* screen is displayed from which the user may enter keywords functionally related to the action of the desired UNIX system commands they wish to have identified. A user may enter keywords and receive a list of UNIX system commands whose functional attributes match those in the keyword list, or may exit to the shell by typing q (for "quit"). For example, if you wish to print the contents of a file, enter the keywords "print" and "file". The *locate* command would then print the names of all commands related to these keywords.

Keywords may also be entered directly from the shell, as shown above. In this case, the initial screen is not displayed, and the resulting command list is printed.

More detailed information on a command in the list produced by *locate* can be obtained by accessing the *usage* module of the UNIX System *help* Facility. Access is made by entering the appropriate menu choice after the command list is displayed.

From any screen in the facility, a user may execute a command via the shell (*sh(1)*) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the *help* facility scrolls the data that is presented to the user. If a user prefers to have the screen clear before printing the data (non-scrolling), a variable must be defined in the user's .profile file called SCROLL. The variable SCROLL must be set to no and exported for non-scrolling to occur. If the user later decides that scrolling is desired, the variable SCROLL must be set to yes or deleted from the user's .profile file.

Further information on the UNIX System *help* Facility can be found on the *help(1)*, *usage(1)*, *starter(1)*, and *glossary(1)* manual pages.

SEE ALSO

glossary(1), *help(1)*, *sh(1)*, *starter(1)*, *usage(1)*.
term(5) in the *3B2 Computer System Programmer Reference Manual*.

WARNINGS

If the TERM variable is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term(5)*. The *help* facility assumes that tabs are set on the user's terminal.

NAME

login — sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-d* to indicate an “end-of-file.” (See *How to Get Started* at the beginning of this volume for instructions on how to dial up initially.)

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

exec login

from the initial shell.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second “dialup” password. This will occur only for dial-up connections, and will be prompted by the message “dialup password:”. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh(1)*) is initialized, and the file *.profile* in the working directory is executed, if it exists. These specifications are found in the */etc/passwd* file entry for the user. The name of the command interpreter is — followed by the last component of the interpreter’s path name (i.e., *-sh*). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used. If this field is “*”, then the named directory becomes the root directory, the starting point for path searches for path names beginning with a /. At that point *login* is re-executed at the new level which must have its own root structure, including */etc/login* and */etc/passwd*.

The basic *environment* is initialized to:

```
HOME=your-login-directory
PATH=:bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
```

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

Ln=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables PATH and

SHELL cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/ <i>your-name</i>	mailbox for user <i>your-name</i>
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), *newgrp(1)*, *sh(1)*.
passwd(4), *profile(4)*, *environ(5)* in the *3B2 Computer System Programmer Reference Manual*.
su(1M) in the *3B2 Computer System Administration Utilities Guide*.

DIAGNOSTICS

Login incorrect if the user name or the password cannot be matched.
No shell, cannot open password file, or no directory: consult a UNIX system programming counselor.
No utmp entry. You must exec "login" from the lowest level "sh": if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

NAME

logname — get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable \$LOGNAME, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1).

environ(5), logname(3X) in the *3B2 Computer System Programmer Reference Manual*.

NAME

ls — list contents of directory

SYNOPSIS

ls [**-RadCxmlnogrtpFbqisf**] [names]

DESCRIPTION

For each directory argument, **ls** lists the contents of the directory; for each file argument, **ls** repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **-C** and **-x** options enable multi-column formats, and the **-m** option enables stream output format in which files are listed across the page, separated by commas. In order to determine output formats for the **-C**, **-x**, and **-m** options, **ls** uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the **terminfo** database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

There are many options:

- R** Recursively list subdirectories encountered.
- a** List all entries; usually entries whose names begin with a period (.) are not listed.
- d** If an argument is a directory, list only its name (not its contents); often used with **-l** to get the status of a directory.
- C** Multi-column output with entries sorted down the columns.
- x** Multi-column output with entries sorted across rather than down the page.
- m** Stream output format.
- l** List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
- n** The same as **-l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.
- o** The same as **-l**, except that the group is not printed.
- g** The same as **-l**, except that the owner is not printed.
- r** Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
- t** Sort by time modified (latest first) instead of by name.
- u** Use time of last access instead of last modification for sorting (with the **-t** option) or printing (with the **-l** option).
- c** Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**-t**) or printing (**-l**).
- p** Put a slash (/) after each filename if that file is a directory.
- F** Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

- b** Force printing of non-graphic characters to be in the octal \ddd notation.
- q** Force printing of non-graphic characters in file names as the character (?).
- i** For each file, print the i-number in the first column of the report.
- s** Give size in blocks, including indirect blocks, for each entry.
- f** Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

The mode printed under the **-l** option consists of 10 characters that are interpreted as follows:

The first character is:

- d** if the entry is a directory;
- b** if the entry is a block special file;
- c** if the entry is a character special file;
- p** if the entry is a fifo (a.k.a. "named pipe") special file;
- if the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

The permissions are indicated as follows:

- r** if the file is readable;
- w** if the file is writable;
- x** if the file is executable;
- if the indicated permission is *not* granted.

The group-execute permission character is given as **s** if the file has set-group-ID mode; likewise, the user-execute permission character is given as **s** if the file has set-user-ID mode. The last character of the mode (normally **x** or **-**) is **t** if the 1000 (octal) bit of the mode is on; see *chmod(1)* for the meaning of this mode. The indications of set-ID and 1000 bits of the mode are capitalized (**S** and **T** respectively) if the corresponding execute permission is *not* set.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	to get user IDs for ls -l and ls -o .
/etc/group	to get group IDs for ls -l and ls -g .
/usr/lib/terminfo/*	to get terminal information.

SEE ALSO

chmod(1), *find(1)*.

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

pdp11, u3b, u3b2, u3b5, vax — provide truth value about your processor type

SYNOPSIS

pdp11

u3b

u3b2

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

pdp11 True if you are on a PDP-11/45 or PDP-11/70.

u3b True if you are on a 3B20 computer.

u3b2 True if you are on a 3B2 computer.

u3b5 True if you are on a 3B5 computer.

vax True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within *make(1)* makefiles and shell procedures to increase portability.

SEE ALSO

sh(1), *test(1)*, *true(1)*.

make(1) in the *3B2 Computer System Extended Software Generation System Utilities*.

NAME

mail, rmail — send mail to users or read mail

SYNOPSIS

```
mail [ -epqr ] [ -f file ]
mail [ -t ] persons
rmail [ -t ] persons
```

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
-	Go back to previous message.
s [files]	Save message in the named <i>files</i> (<i>mbox</i> is default).
w [files]	Save message, without its header, in the named <i>files</i> (<i>mbox</i> is default).
m [persons]	Mail the message to the named <i>persons</i> (<i>yourself</i> is default).
q	Put undeleted mail back in the <i>mailfile</i> and stop.
EOT (control-d)	Same as q.
x	Put all mail back in the <i>mailfile</i> unchanged and stop.
!command	Escape to the shell to do <i>command</i> .
*	Print a command summary.

The optional arguments alter the printing of the mail:

-e	causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
-p	causes all mail to be printed without prompting for disposition.
-q	causes <i>mail</i> to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
-r	causes messages to be printed in first-in, first-out order.
-ffile	causes <i>mail</i> to use <i>file</i> (e.g., <i>mbox</i>) instead of the default <i>mailfile</i> .
-t	causes the message to be preceded by all <i>persons</i> the <i>mail</i> is sent to.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person*'s *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. A *person* is usually a user name recognized by *login(1)*. If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of **dead.letter**.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one

machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

Mail may be set to a recipient on a remote system if you have the Basic Networking Utilities installed. Prefix *person* by the system name and exclamation mark. Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying *a!b!cde* as a recipient's name causes the message to be sent to user *b!cde* on system *a*. System *a* will interpret that destination as a request to send the message to user *cde* on system *b*. This might be useful, for instance, if the sending system can access system *a* but not system *b*, and system *a* has access to system *b*. *Mail* will not use *uucp* if the remote system is the local system name (i.e., *localsystem!user*).

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/ <i>user</i>	incoming mail for <i>user</i> ; i.e., the <i>mailfile</i>
\$HOME/mbox	saved mail
\$MAIL	variable containing path name of <i>mailfile</i>
/tmp/ma*	temporary file
/usr/mail/*.*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), *mailx*(1), *write*(1).
Basic Networking Utilities.

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a p.

NAME

mailx — interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the system *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages remain in this file until forcibly removed.

On the command line, *options* start with a dash (—) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

-e	Test for presence of mail. <i>Mailx</i> prints nothing and exits with a successful return code if there is mail to read.
-f [filename]	Read messages from <i>filename</i> instead of <i>mailbox</i> . If no <i>filename</i> is specified, the <i>mbox</i> is used.
-F	Record the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
-h number	The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops.
-H	Print header summary only.
-i	Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
-n	Do not initialize from the system default <i>Mailx.rc</i> file.
-N	Do not print initial header summary.
-r address	Pass <i>address</i> to network delivery software. All tilde commands are disabled.
-s subject	Set the Subject header field to <i>subject</i> .
-u user	Read <i>user</i> 's <i>mailbox</i> . This is only effective if <i>user</i> 's <i>mailbox</i> is not read protected.
-U	Convert <i>uucp</i> style addresses to internet standards. Overrides the "conv" environment variable.

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by

beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the *set* and *unset* commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp(1)* for recording outgoing mail on paper. Alias groups are set by the alias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, print is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a 'current' message, marked by a '>' in the header summary. Many commands take an optional list of messages (*msglist*) to operate on, which defaults to the current message. A *msglist* is a list of message specifications separated by spaces, which may include:

n	Message number n .										
:	The current message.										
:	The first undeleted message.										
\$	The last message.										
*	All messages.										
n-m	An inclusive range of message numbers.										
user	All messages from user .										
/string	All messages with string in the subject line (case ignored).										
: c	All messages of type c , where c is one of: <table border="0" style="margin-left: 20px;"> <tr> <td>d</td><td>deleted messages</td></tr> <tr> <td>n</td><td>new messages</td></tr> <tr> <td>o</td><td>old messages</td></tr> <tr> <td>r</td><td>read messages</td></tr> <tr> <td>u</td><td>unread messages</td></tr> </table>	d	deleted messages	n	new messages	o	old messages	r	read messages	u	unread messages
d	deleted messages										
n	new messages										
o	old messages										
r	read messages										
u	unread messages										

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh(1)*). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* reads commands from a system-wide file (*/usr/lib/mailx/mailx.rc*) to initialize certain parameters, then from a private start-up file (*\$HOME/.mailrc*) for personalized variables. Most regular commands are legal inside start-up files, the most common use being to set up initial display options and alias lists. The following commands are not legal in the start-up file: !, Copy, edit, followup, Followup, hold, mail, preserve, reply, Reply, shell, and visual. Any errors in the start-up file cause the remaining

lines in the file to be ignored.

COMMANDS

The following is a complete list of *mailx* commands:

!shell-command

Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).

comment

Null command (comment). This may be useful in *.mailrc* files.

=

Print the current message number.

?

Prints a summary of commands.

alias alias name ...

group alias name ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

alternates name ...

Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, *alternates* prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).

cd [directory]

chdir [directory]

Change directory. If *directory* is not specified, \$HOME is used.

copy [filename]

copy [msglist] filename

Copy messages to the file without marking the messages as saved. Otherwise equivalent to the *save* command.

Copy [msglist]

Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the *Save* command.

delete [msglist]

Delete messages from the *mailbox*. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).

discard [header-field ...]

ignore [header-field ...]

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The Print and Type commands override this command.

dp [msglist]

dt [msglist]

Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a *delete*

command followed by a print command.

echo *string* ...

Echo the given strings (like *echo(1)*).

edit [*msglist*]

Edit the given messages. The messages are placed in a temporary file and the "EDITOR" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES). Default editor is *ed(1)*.

exit

xit

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also *quit*).

file [*filename*]

folder [*filename*]

Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

% the current *mailbox*.

%*user* the *mailbox* for *user*.

the previous file.

& the current *mbox*.

Default file is the current *mailbox*.

folders

Print the names of the files in the directory set by the "folder" variable (see ENVIRONMENT VARIABLES).

followup [*message*]

Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the "record" variable, if set. See also the Followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

Followup [*msglist*]

Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the followup, Save, and Copy commands and "outfolder" (ENVIRONMENT VARIABLES).

from [*msglist*]

Prints the header summary for the specified messages.

group *alias name* ...

alias *alias name* ...

Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the *.mailrc* file.

headers [*message*]

Prints the page of headers which includes the message specified. The "screen" variable sets the number of headers per page (see ENVIRONMENT VARIABLES). See also the *z* command.

help

Prints a summary of commands.

hold [msglist]**preserve [msglist]**

Holds the specified messages in the *mailbox*.

if \$r

mail-commands

else

mail-commands

endif

Conditional execution, where *s* will execute following *mail-commands*, up to an **else** or **endif**, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the *.mailrc* file.

ignore header-field ...**discard header-field ...**

Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The Print and Type commands override this command.

list

Prints all commands available. No explanation is given.

mail name ...

Mail a message to the specified users.

mbox [msglist]

Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [message]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [msglist] [shell-command]**| [msglist] [shell-command]**

Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [msglist]**hold [msglist]**

Preserve the specified messages in the *mailbox*.

Print [*msglist*]**Type** [*msglist*]

Print the specified messages on the screen, including all header fields.
Overrides suppression of fields by the ignore command.

print [*msglist*]**type** [*msglist*]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

quit

Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]**Respond** [*msglist*]

Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [*message*]**respond** [*message*]

Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [*msglist*]

Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy, followup, and Followup commands and "outfolder" (ENVIRONMENT VARIABLES).

save [*filename*]**save** [*msglist*] *filename*

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the exit and quit commands).

set**set** *name***set** *name*=*string***set** *name*=*number*

Define a variable called *name*. The variable may be given a null, string, or numeric value. Set by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the *mailx* variables.

shell

Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [msglist]

Print the size in characters of the specified messages.

source filename

Read commands from the given file and return to command mode.

top [msglist]

Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [msglist]

Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox* upon normal termination. See *exit* and *quit*.

Type [msglist]**Print [msglist]**

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the *ignore* command.

type [msglist]**print [msglist]**

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is *pg*(1) (see ENVIRONMENT VARIABLES).

undelete [msglist]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).

unset name ...

Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version

Prints the current version and release date.

visual [msglist]

Edit the given messages with a screen editor. The messages are placed in a temporary file and the "VISUAL" variable is used to get the name of the editor (see ENVIRONMENT VARIABLES).

write [msglist] filename

Write the given messages on the specified file, minus the header and trailing blank line. Otherwise equivalent to the *save* command.

xit**exit**

Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also *quit*).

z[+|-]

Scroll the header display forward or backward one screen-full. The number of headers displayed is set by the "screen" variable (see ENVIRONMENT VARIABLES).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for changing this special character.

~! shell-command

Escape to the shell.

~.

Simulate end of file (terminate message input).

~: mail-command**~- mail-command**

Perform the command-level request. Valid only when sending a message while reading mail.

~?

Print a summary of tilde escapes.

~A

Insert the autograph string "Sign" into the message (see ENVIRONMENT VARIABLES).

~a

Insert the autograph string "sign" into the message (see ENVIRONMENT VARIABLES).

~b name ...

Add the *names* to the blind carbon copy (Bcc) list.

~c name ...

Add the *names* to the carbon copy (Cc) list.

~d

Read in the *dead.letter* file. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~e

Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT VARIABLES).

~f [msglist]

Forward the specified messages. The messages are inserted into the message, without alteration.

~h

Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~i string

Insert the value of the named variable into the text of the message. For example, ~A is equivalent to '~i Sign.'

~m [msglist]

Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~p

Print the message being entered.

~q

Quit from input mode by simulating an interrupt. If the body of the message is not null, the partial message is saved in *dead.letter*. See "DEAD" (ENVIRONMENT VARIABLES) for a description of this file.

~r filename

~< filename

~< !shell-command

Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~s string ...

Set the subject line to *string*.

~t name ...

Add the given *names* to the To list.

~v

Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w filename

Write the partial message onto the given file, without the header.

~x

Exit as with ***~q*** except the message is not saved in *dead.letter*.

| shell-command

Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*.

HOME=*directory*

The user's base of operations.

MAILRC=*filename*

The name of the start-up file. Default is \$HOME/.mailrc.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the set command at any time. The unset command may be used to erase variables.

allnet

All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to

behave similarly. Default is **noallnet**. See also the **alternates** command and the "metoo" variable.

append

Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend**.

askcc

Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub

Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint

Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang

Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=shell-command

Set the default command for the **pipe** command. No default value.

conv=conversion

Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also "sendmail" and the **-U** command line option.

crt=number

Pipe messages having more than *number* lines through the command specified by the value of the "PAGER" variable (*pg*(1) by default). Disabled by default.

DEAD=filename

The name of the file in which to save partial letters in case of untimely interrupt or delivery errors. Default is \$HOME/dead.letter.

debug

Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot

Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=shell-command

The command to run when the **edit** or **~e** command is used. Default is *ed*(1).

escape=c

Substitute *c* for the ~ escape character.

folder=directory

The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), \$HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, "folder" must be an exported *sh* environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header

Enable printing of the header summary when entering *mailx*. Enabled by default.

hold

Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore

Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof

Ignore end-of-file during message input. Input must be terminated by a period(.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also "dot" above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*

The name of the file to save messages which have been read. The exit command overrides this function, as does saving the message explicitly in another file. Default is \$HOME/mbox.

metoo

If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*

The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder

Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is **nooutfolder**. See "folder" above and the Save, Copy, followup, and Followup commands.

page

Used with the pipe command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*

The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*

Set the *command mode* prompt to *string*. Default is "? ".

quiet

Refrain from printing the opening message and version when entering mailx. Default is **noquiet**.

record=*filename*

Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save

Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*

Sets the number of lines in a screen—full of headers for the **headers** command.

sendmail=*shell-command*

Alternate command for delivering messages. Default is *mail*(1).

sendwait

Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*

The name of a preferred command interpreter. Default is *sh*(1).

showto

When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*

The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also **~i** (TILDE ESCAPES)).

Sign=*string*

The variable inserted into the text of a message when the **~A** command is given. No default (see also **~i** (TILDE ESCAPES)).

toplines=*number*

The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*

The name of a preferred screen editor. Default is *vi*(1).

FILES

\$HOME/.mailrc	personal start-up file
\$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO

mail(1), *pg*(1), *ls*(1).
Basic Networking Utilities.

BUGS

Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**. The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a “.” are treated as the end of the message by *mail*(1) (the standard mail delivery program).

NAME

mesg — permit or deny messages

SYNOPSIS

mesg [n] [y]

DESCRIPTION

Mesg with argument **n** forbids messages via *write(1)* by revoking non-user write permission on the user's terminal. *Mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mkdir — make a directory

SYNOPSIS

mkdir dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777 (possibly altered by *umask(1)*). Standard entries, ., for the directory itself, and .., for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

sh(1), rm(1), umask(1).

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made; otherwise, it prints a diagnostic and returns non-zero.

NAME

newform — change the format of a text file

SYNOPSIS

newform [**-s**] [**-itabspec**] [**-otabspec**] [**-bn**] [**-en**] [**-pn**] [**-an**] [**-f**] [**-cchar**] [**-ln**] [*files*]

DESCRIPTION

Newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **-s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like “**-e15 -l60**” will yield results different from “**-l60 -e15**”. Options are applied to all *files* on the command line.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform -s -i -l -a -e file-name

-itabspec Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs(1)*. In addition, *tabspec* may be **--**, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec(4)*). If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** expects no tabs; if any are found, they are treated as **-1**.

-otabspec Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **-itabspec**. If no *tabspec* is given, *tabspec* defaults to **-8**. A *tabspec* of **-0** means that no spaces will be converted to tabs on output.

-bn Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **-ln**). Default is to truncate the number of characters necessary to obtain the effective line length.

The default value is used when **-b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

`newform -l1 -b7 file-name`

-en Same as **-bn** except that characters are truncated from the end of the line.

-pn Prefix *n* characters (see **-ck**) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

-an Same as **-pn** except characters are appended to the end of a line.

-f Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **-o** option. If no **-o** option is specified, the line which is printed will contain the default specification of **-8**.

-ck Change the prefix/append character to *k*. Default character for *k* is a space.

-ln Set the effective line length to *n* characters. If *n* is not entered, **-l** defaults to 72. The default line length without the **-l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **-i** to expand tabs to spaces).

The **-l1** must be used to set the effective line length shorter than any existing line in the file so that the **-b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...

not -s format

can't open file

internal line too long

tabspec in error

tabspec indirection illegal

0 — normal execution

1 — for any error

Newform was called with a bad option.

There was no tab on one line.

Self-explanatory.

A line exceeds 512 characters after being expanded in the internal work buffer.

A tab specification is incorrectly formatted, or specified tab stops are not ascending.

A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

SEE ALSO

csplit(1), tabs(1).

fspec(4) in the *3B2 Computer System Programmer Reference Manual*.

BUGS

Newform normally only keeps track of physical characters; however, for the **-i** and **-o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

Newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **-i--** or **-o--**).

If the **-f** option is used, and the last **-o** option specified was **-o--**, and was preceded by either a **-o--** or a **-i--**, the tab specification format line will be incorrect.

NAME

news — print news items

SYNOPSIS

news [**-a**] [**-n**] [**-s**] [items]

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory */usr/news*.

When invoked without arguments, *news* prints the contents of all current files in */usr/news*, most recent first, with each preceded by an appropriate header. *News* stores the “currency” time as the modification date of a file named *.news_time* in the user’s home directory (the identity of this directory is determined by the environment variable *\$HOME*); only files more recent than this currency time are considered “current.”

- a** option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.
- n** option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.
- s** option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one’s *.profile* file, or in the system’s */etc/profile*.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
*/usr/news/**
\$HOME/.news_time

SEE ALSO

profile(4), *environ(5)* in the *3B2 Computer System Programmer Reference Manual*.

NAME

nice — run a command at low priority

SYNOPSIS

nice [*—increment*] *command* [*arguments*]

DESCRIPTION

Nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **—10**.

SEE ALSO

nohup(1).

nice(2) in the *3B2 Computer System Programmer Reference Manual*.

DIAGNOSTICS

Nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl — line numbering filter

SYNOPSIS

```
nl [-htype] [-btype] [-ftype] [-vstart#] [-iincr] [-p] [-lnum] [-ssep]
    [-wwidth] [-nformat] [-ddelim] file
```

DESCRIPTION

Nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

Nl views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

<i>Line contents</i>	<i>Start of</i>
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

- btype** Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:
 - a number all lines
 - t number lines with printable text only
 - n no line numbering
 - pstring number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is t (text lines numbered).

- ftype** Same as **-btype** except for footer. Default for logical page footer is n (no lines numbered).
- vstart#** *Start#* is the initial value used to number logical page lines. Default is 1.
- iincr** *Incr* is the increment value used to number logical page lines. Default is 1.
- p** Do not restart numbering at logical page delimiters.
- lnum-** *Num* is the number of blank lines to be considered as one. For example, -l2 results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is 1.
- ssep** *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

- wwidth** *Width* is the number of characters to be used for the line number. Default *width* is 6.
- nformat** *Format* is the line numbering format. Recognized values are: **l**, left justified, leading zeroes suppressed; **r**, right justified, leading zeroes suppressed; **rz**, right justified, leading zeroes kept. Default *format* is **r** (right justified).
- dxx** The delimiter characters specifying the start of a logical page section may be changed from the default characters (:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **-d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are **!+**.

SEE ALSO

pr(1).

NAME

nohup — run a command immune to hangups and quits

SYNOPSIS

nohup *command* [arguments]

DESCRIPTION

Nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to *nohup.out*. If *nohup.out* is not writable in the current directory, output is redirected to \$HOME/nohup.out.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh(1)*):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), *nice(1)*, *sh(1)*,
signal(2) in the 3B2 Computer System Programmer Reference Manual.

WARNINGS

nohup command1; command2 *nohup* applies only to *command1*
nohup (command1; command2) is syntactically incorrect.

Be careful of where standard error is redirected. The following command may put error messages on tape, making it unreadable:

```
nohup cpio -o <list >/dev/rmt/1m&
while
    nohup cpio -o <list >/dev/rmt/1m 2>errors&
```

puts the error messages into file *errors*.

NAME

od - octal dump

SYNOPSIS

od [-bcdosx] [file] [[+] offset[.][b]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **-o** is default. The meanings of the format options are:

- b** Interpret bytes in octal.
- c** Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
- d** Interpret words in unsigned decimal.
- o** Interpret words in octal.
- s** Interpret 16-bit words in signed decimal.
- x** Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

NAME

pack, pcat, unpack — compress and expand files

SYNOPSIS

```
pack [ - ] [ -f ] name ...
pcat name ...
unpack name ...
```

DESCRIPTION

Pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. The *-f* option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

Pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the *-* argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of *-* in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each *.z* file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

Pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

- the file appears to be already packed;
- the file name has more than 12 characters;
- the file has links;
- the file is a directory;
- the file cannot be opened;
- no disk storage blocks will be saved by packing;
- a file called *name.z* already exists;
- the *.z* file cannot be created;
- an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended *.z* extension. Directories cannot be compressed.

Pcat does for packed files what *cat(1)* does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name.z* use:

```
pcat name.z
or just:
pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name.z* (without destroying *name.z*) use the command:

```
pcat name > nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:
the file name (exclusive of the .z) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

Unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name.z* (or just *name*, if *name* ends in .z). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the .z suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

- a file with the “unpacked” name already exists;
- if the unpacked file cannot be created.

SEE ALSO

cat(1).

NAME

passwd — change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes or installs a password associated with the login *name*.

Ordinary users may change only the password which corresponds to their login *name*.

Passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd(4)*.

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id(1)*, and *su(1)*. Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

login(1).

crypt(3C), *passwd(4)* in the *3B2 Computer System Programmer Reference Manual*.

id(1M), *su(1M)* in the *3B2 Computer System Administration Utilities Guide*.

NAME

paste — merge same lines of several files or subsequent lines of one file

SYNOPSIS

```
paste file1 file2 ...
paste -dlist file1 file2...
paste -s [-dlist] file1 file2 ...
```

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat(1)* which concatenates vertically, i.e., one file after the other. In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if **-** is used in place of a file name.

The meanings of the options are:

- d** Without this option, the new-line characters of each but the last file (or last line in case of the **-s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
- list** One or more characters immediately following **-d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **-s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line), **\t** (tab), **** (backslash), and **\0** (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use **-d"\\\\\"**).
- s** Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **-d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d" " -
ls | paste ----
paste -s -d"\t\n" file
```

```
list directory in one column
list directory in four columns
combine pairs of lines into lines
```

SEE ALSO

cut(1), *grep(1)*, *pr(1)*.

DIAGNOSTICS

line too long

Output lines are restricted to 511 characters.

too many files

Except for **-s** option, no more than 12 input files may be specified.

NAME

pg — file perusal filter for CRTs

SYNOPSIS

pg [**-number**] [**-p string**] [**-cefnsl**] [**+linenumber**] [**+/pattern/**] [**files...**]

DESCRIPTION

The **pg** command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name — and/or NULL arguments indicate that **pg** should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, **pg** scans the *terminfo* (see the *3B2 Computer System Terminal Information Utilities Guide*) data base for the terminal type specified by the environment variable TERM. If TERM is not defined, the terminal type **dumb** is assumed.

The command line options are:

-number

An integer specifying the size (in lines) of the window that **pg** is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

-p string

Causes **pg** to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

-c

Home the cursor and clear the screen before displaying each page. This option is ignored if **clear screen** is not defined for this terminal type in the *terminfo* (see the *3B2 Computer System Terminal Information Utilities Guide*) data base.

-e

Causes **pg** *not* to pause at the end of each file.

-f

Normally, **pg** splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The **-f** option inhibits **pg** from splitting lines.

-n

Normally, commands must be terminated by a <newline> character. This option causes an automatic end of command as soon as a command letter is entered.

-s

Causes **pg** to print all messages and prompts in standout mode (usually inverse video).

+linenumber

Start up at *linenumber*.

+/pattern/

Start up at the first line containing the regular expression pattern.

The responses that may be typed when **pg** pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be

displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1) <newline> or <blank>

This causes one page to be displayed. The address is specified in pages.

(+1) l With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) d or ^D

Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or ^L Typing a single period causes the current page of text to be redisplayed.

\$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed(1)* are available. They must always be terminated by a <newline>, even if the *-n* option is specified.

i/pattern/

Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^pattern^

i?pattern?

Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

in Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

ip Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

iw Display another window of text. If *i* is present, set the window size to *i*.

s filename

Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a <newline>, even if the *-n* option is specified.

h Help by displaying an abbreviated summary of available commands.

q or **Q** Quit *pg*.

!command

Command is passed to the shell, whose name is taken from the **SHELL** environment variable. If this is not available, the default shell is used. This command must always be terminated by a <newline>, even if the **-n** option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the **z** and **f** commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

1027.sp40u

/usr/lib/terminfo/*

Terminal information data base

/tmp/pg*

Temporary file when input is from a pipe

SEE ALSO

ed(1), *grep*(1).

3B2 Computer System Terminal Information Utilities Guide.

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

pr — print files

SYNOPSIS

pr [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is **-**, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

The below *options* may appear singly or be combined in any order:

- +k** Begin printing with page *k* (default is 1).
- k** Produce *k*-column output (default is 1). The options **-e** and **-i** are assumed for multi-column output.
- a** Print multi-column output across the page.
- m** Merge and print all files simultaneously, one per column (overrides the **-k**, and **-a** options).
- d** Double-space the output.
- eck** Expand *input* tabs to character positions *k*+1, 2*i+1, 3*i+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
- ick** In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2*i+1, 3*i+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
- nck** Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of **-m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
- wk** Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).
- ok** Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.
- lk** Set the length of a page to *k* lines (default is 66).
- h** Use the next argument as the header to be printed instead of the file name.
- p** Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

- f Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r Print no diagnostic reports on failure to open files.
- t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ... :

```
pr -e9 -t <file1 >file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

ps — report process status

SYNOPSIS

ps [options]

DESCRIPTION

Ps prints certain information about active processes. Without *options*, information is printed about processes associated with the current terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Options using lists as arguments can have the list specified in one of two forms: a list of identifiers separated from one another by a comma, or a list of identifiers enclosed in double quotes and separated from one another by a comma and/or one or more spaces.

The *options* are:

- e** Print information about all processes.
- d** Print information about all processes, except process group leaders.
- a** Print information about all processes, except process group leaders and processes not associated with a terminal.
- f** Generate a *full* listing. (See below for meaning of columns in a full listing).
- l** Generate a *long* listing. See below.
- c corefile** Use the file *corefile* in place of */dev/mem*.
- s swapdev** Use the file *swapdev* in place of */dev/swap*. This is useful when examining a *corefile*; a *swapdev* of */dev/null* will cause the user block to be zeroed out.
- n namelist** The argument will be taken as the name of an alternate system *namelist* file in place of */unix*.
- t termlist** Restrict listing to data about the processes associated with the terminals given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., *tty04*) or if the device's file name starts with *tty*, just the digit identifier (e.g., *04*).
- p proclist** Restrict listing to data about processes whose process ID numbers are given in *proclist*.
- u uidlist** Restrict listing to data about processes whose user ID numbers or login names are given in *uidlist*. In the listing, the numerical user ID will be printed unless the **-f** option is used, in which case the login name will be printed.
- g grplist** Restrict listing to data about processes whose process group leaders are given in *grplist*.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters *f* and *l* indicate the option (*full* or *long*) that causes the corresponding heading to appear; *all* means that the heading always appears. Note that these two options determine only what information is provided for a process; they do *not* determine which processes will be listed.

F	(l)	Flags (octal and additive) associated with the process:
	0	swapped;
	1	in core;
	2	system process;
	4	locked-in core (e.g., for physical I/O);
	10	being swapped;
	20	being traced by another process;
	40	another tracing flag;
	100	3B20 computer: swapin segment expansion; VAX-11/780: text pointer valid;
	200	3B20 computer: process is child (during fork swap); VAX-11/780: process is partially swapped.
S	(l)	The state of the process:
	0	non-existent;
	S	sleeping;
	W	waiting;
	R	running;
	I	intermediate;
	Z	terminated;
	T	stopped;
	X	growing.
UID	(f,l)	The user ID number of the process owner; the login name is printed under the -f option.
PID	(all)	The process ID of the process; it is possible to kill a process if you know this datum.
PPID	(f,l)	The process ID of the parent process.
C	(f,l)	Processor utilization for scheduling.
PRI	(l)	The priority of the process; higher numbers mean lower priority.
NI	(l)	Nice value; used in priority computation.
ADDR	(l)	The memory address of the process (a pointer to the segment table array on the 3B20 computer), if resident; otherwise, the disk address.
SZ	(l)	The size in blocks of the core image of the process.
WCHAN	(l)	The event for which the process is waiting or sleeping; if blank, the process is running.
STIME	(f)	Starting time of the process.
TTY	(all)	The controlling terminal for the process.
TIME	(all)	The cumulative execution time for the process.
CMD	(all)	The command name; the full command name and its arguments are printed under the -f option.

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <defunct>.

Under the -f option, ps tries to determine the command name and arguments given when the process was created by examining memory or the swap area. Failing this, the command name, as it would appear without the -f option, is printed in square brackets.

FILES

/unix	system namelist
/dev/mem	memory
/dev/swap	the default swap device
/etc/passwd	supplies UID information
/etc/ps_data	internal data structure
/dev	searched to find terminal ("tty") names

SEE ALSO

kill(1), nice(1).

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality. Some data printed for defunct processes are irrelevant.

NAME

pwd — working directory name

SYNOPSIS

pwd

DESCRIPTION

Pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

“Cannot open ..” and “Read error in ..” indicate possible file system trouble and should be referred to a UNIX system programming counselor.

NAME

rm, rmdir — remove files or directories

SYNOPSIS

rm [-fri] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with *y* the file is deleted, otherwise the file remains. No questions are asked when the **-f** option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument **-r** has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the **-i** (interactive) option is in effect, *rm* asks whether to delete each file, and, under **-r**, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2) in the *3B2 Computer System Programmer Reference Manual*.

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file .. merely to avoid the antisocial consequences of inadvertently doing something like:

rm -r .*

NAME

sdiff — side-by-side difference program

SYNOPSIS

sdiff [options ...] *file1* *file2*

DESCRIPTION

Sdiff uses the output of *diff(1)* to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

x		y
a		a
b	<	
c	<	
d		d
	>	c

The following options exist:

- w *n*** Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.
- l** Only print the left side of any lines that are identical.
- s** Do not print identical lines.
- o *output*** Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff(1)*, are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

- l** append the left column to the output file
- r** append the right column to the output file
- s** turn on silent mode; do not print identical lines
- v** turn off silent mode
- e l** call the editor with the left column
- e r** call the editor with the right column
- e b** call the editor with the concatenation of left and right
- e** call the editor with a zero length file
- q** exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), *ed(1)*.



NAME

sed — stream editor

SYNOPSIS

sed [-n] [-e script] [-f sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **-f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **-e** option and no **-f** options, the flag **-e** may be omitted. The **-n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a D command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **-n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a \$ that addresses the last line of input, or a context address, i.e., a /*regular expression*/ in the style of *ed*(1) modified thus:

In a context address, the construction \?*regular expression*?, where ? is any character, is identical to /*regular expression*/. Note that in the context address \xabc\xdefx, the second x stands for itself, so that the regular expression is abcxdef.

The escape sequence \n matches a new-line *embedded* in the pattern space.

A period . matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function ! (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

- (1) **a**
text Append. Place *text* on the output before reading the next input line.
- (2) **b label** Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.
- (2) **c**
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
- (2) **d** Delete the pattern space. Start the next cycle.
- (2) **D** Delete the initial segment of the pattern space through the first new-line. Start the next cycle.
- (2) **g** Replace the contents of the pattern space by the contents of the hold space.
- (2) **G** Append the contents of the hold space to the pattern space.
- (2) **h** Replace the contents of the hold space by the contents of the pattern space.
- (2) **H** Append the contents of the pattern space to the hold space.
- (1) **i**
text Insert. Place *text* on the standard output.
- (2) **I** List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.
- (2) **n** Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
- (2) **N** Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
- (2) **p** Print. Copy the pattern space to the standard output.
- (2) **P** Copy the initial segment of the pattern space through the first new-line to the standard output.
- (1) **q** Quit. Branch to the end of the script. Do not start a new cycle.
- (2) **r rfile** Read the contents of *rfile*. Place them on the output before reading the next input line.
- (2) **s/regular expression/replacement/flags** Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see ed(1). *Flags* is zero or more of:
- n** *n*= 1 - 512. Substitute for just the *n* th occurrence of the *regular expression*.
 - g** Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
 - p** Print the pattern space if a replacement was made.
 - w wfile** Write. Append the pattern space to *wfile* if a replacement was made.
- (2) **t label** Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.
- (2) **w wfile** Write. Append the pattern space to *wfile*.
- (2) **x** Exchange the contents of the pattern and hold spaces.
- (2) **y/string1/string2/** Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*

Don't. Apply the *function* (or group, if *function* is {}) only to lines *not* selected by the address(es).

(0) : *label* This command does nothing; it bears a *label* for b and t commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

(0) # If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

setup — initialize system for first user

SYNOPSIS

setup

DESCRIPTION

The *setup* command, which is also accessible as a login by the same name, allows the first user to be established as the "owner" of the machine.

The user is permitted to add the first logins to the system, usually starting with their own.

The user can then protect the system from unauthorized modification of the machine configuration and software by giving passwords to the administrative and maintenance functions. Normally, the first user of the machine enters this command through the setup login, which initially has no password, and then gives passwords to the various functions in the system. Any that the user leaves without password protection can be exercised by anyone.

The user can then give passwords to system logins such as "root", "bin", etc. (*provided they do not already have passwords*). Once given a password, each login can only be changed by that login or "root".

The user can then set the date, time and time zone of the machine.

The user can then set the node name of the machine.

SEE ALSO

password(1).

DIAGNOSTICS

The *passwd(1)* command complains if the password provided does not meet its standards.

WARNING

If the setup login is not under password control, anyone can put passwords on the other functions.

NAME

sh, rsh — shell, the standard/restricted command programming language

SYNOPSIS

```
sh [ -acefhiknrstuvx ] [ args ]
rsh [ -acefhiknrstuvx ] [ args ]
```

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, -, \$, and !.

Commands

A *simple-command* is a sequence of non-blank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec(2)*). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal(2)* for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ^). The standard output of each command but the last is connected by a *pipe(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for name [in word ...] do list done

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case word in [pattern [| pattern] ...] list ;;] ... esac

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation*) except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else list** is executed. If no **else list** or **then list** is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(list)

Execute *list* in a sub-shell.

{list;}

list is simply executed.

name **O** **{list;}**

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if **then** **else** **elif** **fi** **case** **esac** **for** **while** **until** **do** **done** { }

Comments

A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution

The standard output from a command enclosed in a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution

The character \$ is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters (also known as variables) may be assigned values by writing:

name = value [*name = value*] ...

Pattern-matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

\${parameter}

The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with \$1, are substituted (separated by spaces). Parameter \$0 is set from argument zero when the shell is invoked.

\${parameter:-word}

If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

\${parameter:=word}

If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

`$(parameter[:word])`

If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

`$(parameter:+word)`

If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, `pwd` is executed only if `d` is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- # The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the `set` command.
- ? The decimal value returned by the last synchronously executed command.
- \$ The process number of this shell.
- ! The process number of the last background command invoked.

The following parameters are used by the shell:

- HOME** The default argument (home directory) for the `cd` command.
- PATH** The search path for commands (see *Execution* below). The user may not change `PATH` if executing under `rsh`.

CDPATH

The search path for the `cd` command.

- MAIL** If this parameter is set to the name of a mail file and the `MAILPATH` parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK

This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the `MAILPATH` or `MAIL` parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH

A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "\$".

PS2 Secondary prompt string, by default ">".

IFS Internal field separators, normally space, tab, and new-line.

SHACCT

If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see *Environment* below) for this name. If it is found and there is an 'r' in the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to PATH, PS1, PS2, MAILCHECK and IFS. HOME and MAIL are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments (" " or "") are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

File Name Generation

Following substitution, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

- * Matches any string, including the null string.
- ? Matches any single character.
- [...] Matches any one of the enclosed characters. A pair of characters separated by - matches any character lexically between the pair, inclusive. If the first character following the opening "[" is a "?" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & () | ^ < > new-line space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair \new-line is ignored. All characters enclosed between a pair of single quote marks (''), except a single quote, are quoted. Inside double quote marks (""), parameter and command substitution occurs and \ quotes the characters \, ` , " , and \$. "\$*" is equivalent to "\$1 \$2 ...", whereas "\$@" is equivalent to "\$1" "\$2"

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a command. If at any time a new-line is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of PS2) is issued.

Input/Output

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are not passed on to the invoked command; substitution occurs before *word* or *digit* is used:

- <*word* Use file *word* as standard input (file descriptor 0).
- >*word* Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.
- >>*word* Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
- <<[-]*word* The shell input is read up to a line that is the same as *word*, or to an end-of-file. The resulting document becomes the standard input. If any character of *word* is quoted, no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unesapeded) \new-line is ignored, and \ must be used to quote the characters

\, \$, ` , and the first character of *word*. If - is appended to <<, all leading tabs are stripped from *word* and from the document.

- | | |
|------------------------|--|
| <& digit | Use the file associated with file descriptor <i>digit</i> as standard input. Similarly for the standard output using >& <i>digit</i> . |
| <& - | The standard input is closed. Similarly for the standard output using >& -. |

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

```
... 1>xxx 2>&1
```

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

If a command is followed by & the default standard input for the command is the empty file /dev/null. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment

The *environment* (see *environ(5)*) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd                                and  
(export TERM; TERM=450; cmd)
```

are equivalent (as far as the execution of *cmd* is concerned).

If the -k flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c  
set -k  
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters \$1, \$2, are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, /bin, and /usr/bin, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the PATH variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. file Read and execute commands from *file* and return. The search path specified by PATH is used to find the directory containing *file*.

break [n]

Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [n]

Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [arg]

Change the current directory to *arg*. The shell parameter HOME is the default *arg*. The shell parameter CDPATH defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The cd command may not be executed by *rsh*.

echo [*arg* ...]
Echo arguments. See *echo(1)* for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, a list of all names that are exported in this shell is printed. Function names may *not* be exported.

hash [-r] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. There are certain situations which require that the stored location of a command be recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to exec *newgrp arg* See *newgrp(1)* for usage and description.

pwd Print the current working directory. See *pwd(1)* for usage and description.

read [*name* ...]
One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [--aefhknntuvx [arg ...]]

- a** Mark variables which are modified or created for export.
- e** Exit immediately if a command exits with a non-zero exit status.
- f** Disable file name generation
- h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- n** Read commands but do not execute them.
- t** Exit after reading and executing one command.
- u** Treat unset variables as an error when substituting.
- v** Print shell input lines as they are read.
- x** Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting \$1 to -.

Using + rather than - causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in \$-. The remaining arguments are positional parameters and are assigned, in order, to \$1, \$2, If no arguments are given the values of all names are printed.

shift [n]

The positional parameters from \$n+1 ... are renamed \$1 If n is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See *test(1)* for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [arg] [n] ...

The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [name ...]

For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [-fp] [n]

imposes a size limit of *n*

- f** imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.
- p** changes the pipe size to *n* (UNIX system/RT only).

If no option is given, -f is assumed.

umask [nnn]

The user file-creation mask is set to *nnn* (see *umask(2)*). If *nnn* is omitted, the current value of the mask is printed.

unset [name ...]

For each *name*, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK and IFS cannot be unset.

wait [n]

Wait for the specified process and report its termination status. If *n* is not given all currently active child processes are waited for and the return code is zero.

Invocation

If the shell is invoked through *exec(2)* and the first character of argument zero is *-*, commands are initially read from /etc/profile and from \$HOME/.profile, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as /bin/sh. The flags below are interpreted by the shell on invocation only; Note that unless the *-c* or *-s* flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c string If the *-c* flag is present commands are read from *string*.

-s If the *-s* flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

-i If the *-i* flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that kill 0 does not kill an interactive shell) and INTERRUPT is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.

-r If the *-r* flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the set command above.

Rsh Only

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd(1)*),

setting the value of \$PATH,

specifying path or command names containing /,

redirecting output (> and >>).

The restrictions above are enforced after .profile is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the .profile has complete control over user actions, by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., /usr/rbin) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
\$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), echo(1), env(1), login(1), newgrp(1), pwd(1), test(1), umask(1).
dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2) in the *3B2 Computer System Programmer Reference Manual*.

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

NAME

shl — shell layer manager

SYNOPSIS

shl

DESCRIPTION

Shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *Shl* has its own prompt, **>>>**, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt????**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or new-line. Only the first eight characters are significant. The *names* (1) through (7) cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [name]

Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form (#) where # is the last digit of the virtual device bound to the layer. The shell prompt variable PS1 is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block name [name ...]

For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **-loblk** within the layer.

delete name [name ...]

For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or ?)

Print the syntax of the *shl* commands.

layers [-l] [name ...]

For each *name*, list the layer name and its process group. The **-l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [name]

Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle Resume the layer that was current before the last current layer.**unblock name [name ...]**

For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty*

option **loblk** within the layer.

quit Exit *shl*. All layers are sent the SIGHUP signal.

name Make the layer referenced by *name* the current layer.

FILES

/dev/sxt??? Virtual tty devices

\$SHELL Variable containing path name of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), *stty(1)*.

ioctl(2), *signal(2)* in the *3B2 Computer System Programmer Reference Manual*.

sxt(7) in the *#b2 Computer System Administration Utilities Guide*.

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

(sleep 105; *command*) &

or to execute a command every so often, as in:

```
while true
do
    command
    sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C) in the *3B2 Computer System Programmer Reference Manual*.

NAME

sort — sort and/or merge files

SYNOPSIS

sort [−cmu] [−o] [−ykmem] [−zrecsz] [−dfiMnr] [−btz] [+pos1
[−pos2]] [files]

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if **−** is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

−c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

−m Merge only, the input files are already sorted.

−u Unique: suppress all but one in each set of lines having equal keys.

−o

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **−o** and *output*.

−ykmem

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **−y0** is guaranteed to start with minimum memory. By convention, **−y** (with no argument) starts with maximum memory.

−zrecsz

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **−c** or **−m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

−d “Dictionary” order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

−f Fold lower case letters into upper case.

−i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

−M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that “JAN” < “FEB” < ... < “DEC”. Invalid fields compare low to “JAN”. The **−M** option implies the **−b** option (see below).

−n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by

arithmetic value. The **-n** option implies the **-b** option (see below). Note that the **-b** option is only effective when restricted sort key specifications are in effect.

- r** Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation **+pos1 -pos2** restricts a sort key to one beginning at *pos1* and ending at *pos2*. The characters at positions *pos1* and *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing **-pos2** means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

- b** Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **-b** option is specified before the first **+pos1** argument, it will be applied to all **+pos1** arguments. Otherwise, the **b** flag may be attached independently to each **+pos1** or **-pos2** argument (see below).
- tx** Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (e.g., *xx* delimits an empty field).

Pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfnr**. A starting position specified by **+m.n** is interpreted to mean the *n*th character in the *m*th field. A missing *.n* means .0, indicating the first character of the *m*th field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*th field; **+m.0b** refers to the first non-blank character in the *m*th field.

A last position specified by **-m.n** is interpreted to mean the *n*th character (including separators) after the last character of the *m*th field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*th field; **-m.1b** refers to the first non-blank in the *m*th field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

```
sort +1 -2 infile
```

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

```
sort -r -o outfile +1.0 -1.2 infile1 infile2
```

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

```
sort -r +1.0b -1.1b infile1 infile2
```

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

```
sort -t: +2n -3 /etc/passwd
```

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **-um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

```
sort -um +2 -3 infile
```

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

DIAGNOSTICS

Comments and exits with non-zero status for various trouble conditions (e.g., when input lines are too long), and for disorder discovered under the **-c** option. When the last line of an input file is missing a new-line character, *sort* appends one, prints a warning message, and continues.

NAME

split — split a file into pieces

SYNOPSIS

split [-n] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if **-** is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1).

NAME

starter — information about the UNIX system for beginning users

SYNOPSIS

[**help**] starter

DESCRIPTION

The UNIX System *help* Facility command *starter* provides five categories of information about the UNIX system to assist new users.

The five categories are:

- commands a new user should learn first
- UNIX system documents important for beginners
- education centers offering UNIX system courses
- local environment information
- on-line teaching aids installed on the UNIX system

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit"). When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the facility, a user may execute a command via the shell (*sh(1)*) by typing a ! and the command to be executed. The screen will be redrawn if the command that was entered was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the *help* facility scrolls the data that is presented to the user. If a user prefers to have the screen cleared before printing the data (non-scrolling), a variable must be defined in the user's .profile file called SCROLL. The variable SCROLL must be set to no and exported for non-scrolling to occur. If the user later decides that scrolling is desired, the variable SCROLL must be set to yes or deleted from the user's .profile file.

Further information on the UNIX System *help* Facility can be found on the *help(1)*, *usage(1)*, *locate(1)*, and *glossary(1)* manual pages.

SEE ALSO

glossary(1), *help(1)*, *locate(1)*, *sh(1)*, *usage(1)*.

term(5) in the *3B2 Computer System Programmer Reference Manual*.

WARNINGS

If the TERM variable is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term(5)*. The *help* facility assumes that tabs are set on the user's terminal.

NAME

stty — set the options for a terminal

SYNOPSIS

stty [-a] [-g] [options]

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options;

-a reports all of the option settings;

-g reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (— parenb)	enable (disable) parity generation and detection.
parodd (— parodd)	select odd (even) parity.
cs5 cs6 cs7 cs8	select character size (see <i>termio(7)</i>).
0	hang up phone line immediately.

110 300 600 1200 1800 2400 4800 9600 exta extb	Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
---	--

hupcl (— hupcl)	hang up (do not hang up) DATA-PHONE® connection on last close.
--------------------------------	--

hup (— hup)	same as hupcl (— hupcl).
cstopb (— cstopb)	use two (one) stop bits per character.
cread (— cread)	enable (disable) the receiver.
clocal (— clocal)	n assume a line without (with) modem control.
loblk (— loblk)	block (do not block) output from a non-current layer.

Input Modes

ignbrk (— ignbrk)	ignore (do not ignore) break on input.
brkint (— brkint)	signal (do not signal) INTR on break.
ignpar (— ignpar)	ignore (do not ignore) parity errors.
parmrk (— parmrk)	mark (do not mark) parity errors (see <i>termio(7)</i>).
inpck (— inpck)	enable (disable) input parity checking.
istrip (— istrip)	strip (do not strip) input characters to seven bits.
inlcr (— inlcr)	map (do not map) NL to CR on input.
igncr (— igncr)	ignore (do not ignore) CR on input.
icrnl (— icrnl)	map (do not map) CR to NL on input.
iucle (— iucle)	map (do not map) upper-case alphabetics to lower case on input.

ixon (— ixon)	enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
------------------------------	--

ixany (— ixany)	allow any character (only DC1) to restart output.
ixoff (— ixoff)	request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (— opost)	post-process output (do not post-process output; ignore all other output modes).
olcuc (— olcuc)	map (do not map) lower-case alphabetics to upper case on output.
onlcr (— onlcr)	map (do not map) NL to CR-NL on output.

ocrnl (-ocrnl)	map (do not map) CR to NL on output.
onocr (-onocr)	do not (do) output CRs at column zero.
onlret (-onlret)	on the terminal NL performs (does not perform) the CR function.
ofill (-ofill)	use fill characters (use timing) for delays.
ofdel (-ofdel)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns (see <i>termio(7)</i>).
nl0 nl1	select style of delay for line-feeds (see <i>termio(7)</i>).
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs (see <i>termio(7)</i>).
bs0 bs1	select style of delay for backspaces (see <i>termio(7)</i>).
ff0 ff1	select style of delay for form-feeds (see <i>termio(7)</i>).
vt0 vt1	select style of delay for vertical tabs (see <i>termio(7)</i>).
Local Modes	
isig (-isig)	enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (-icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase)	canonical (unprocessed) upper/lower-case presentation.
echo (-echo)	echo back (do not echo back) every character typed.
echoe (-echoe)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEd character on many CRT terminals; however, it does <i>not</i> keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (-echok)	echo (do not echo) NL after KILL character.
lfkc (-lfkc)	the same as echok (-echok); obsolete.
echonl (-echonl)	echo (do not echo) NL.
noflsh (-noflsh)	disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (-stwrap)	disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (-stflush)	enable (disable) flush on a synchronous line after every <i>write(2)</i> .
stappl (-stappl)	use application mode (use line mode) on a synchronous line.
Control Assignments	
<i>control-character c</i>	set <i>control-character</i> to <i>c</i> , where <i>control-character</i> is erase, kill, intr, quit, swtch, eof, ctab, min, or time (ctab is used with -stappl; (min and time are used with -icanon; see <i>termio(7)</i>). If <i>c</i> is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., "d" is a CTRL-d); "?" is interpreted as DEL and "^-" is interpreted as undefined.
line i	set line discipline to <i>i</i> ($0 < i < 127$).
Combination Modes	
evenp or parity	enable parenb and cs7.
oddp	enable parenb, cs7, and parodd.
-parity, -evenp, or -oddp	disable parenb, and set cs8.
raw (-raw or cooked)	enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
nl (-nl)	unset (set) icrnl, onlcr. In addition -nl unsets inler, igner, ocrnl, and onlret.

lcase (-lcase)	set (unset) xcase , iuclc , and olcuc .
LCASE (-LCASE)	same as lcase (-lcase).
tabs (-tabs or tab3)	preserve (expand to spaces) tabs when printing.
ek	reset ERASE and KILL characters back to normal # and @.
sane	resets all modes to some reasonable values.
term	set all modes suitable for the terminal type <i>term</i> , where <i>term</i> is one of tty33 , tty37 , vt05 , tn300 , ti700 , or tek .

SEE ALSO**tabs(1)**.**ioctl(2)** in the *3B2 Computer System Programmer Reference Manual*.**termio(7)** in the *3B2 Computer System Administration Utilities Guide*.

NAME

sum — print checksum and block count of a file

SYNOPSIS

sum [-r] file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **-r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

“Read error” is indistinguishable from end of file on most devices; check the block count.

NAME

sysadm — menu interface to do system administration

SYNOPSIS

sysadm [*sub-command*]

DESCRIPTION

This command, when invoked without an argument, presents a menu of system administration sub-commands, from which the user selects. If the optional argument is presented, the named sub-command is run or the named sub-menu use presented.

The **sysadm** command may be given a password. See **admpasswd** in the SUB-COMMANDS section.

SUB-COMMANDS

The following menus of sub-commands are available. (The number of bullets (•) in front of each item indicates the level of the menu or subcommand.)

• **diagnostics**

system diagnostics menu

These subcommands look for and sometimes repair problems in the system. Those subcommands that issue reports allow you to determine if there are detectable problems. Commands that attempt repair are for repair people only. You must know what you are doing!

•• **diskrepair**

advice on repair of built-in disk errors

This subcommand advises you on how to go about repairing errors that occur on built-in disks.

WARNING: Because this is a repair function, it should only be performed by qualified service personnel.

NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

•• **diskreport**

report on built-in disk errors

This subcommand shows you if the system has collected any information indicating that there have been errors while reading the built-in disks. You can request either summary or full reports. The summary report provides sufficient information about disk errors to determine if repair should be attempted. If the message no errors logged is part of the report, then there is probably no damage. If a number of errors is reported, there is damage and you should call for service. The full report gives additional detail for the expert repair person trouble shooting complicated problems.

NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

• **diskmgmt**

disk management menu

The subcommands in this menu provide functions for using removable disks. The subcommands include the ability to format disks, copy disks, and to use disks as mountable file systems. It also contains a

menu of subcommands for handling non-removable media.

•• checkfsys

check a removable disk file system for errors

Checkfsys checks a file system on a removable disk for errors. If there are errors, this procedure attempts to repair them.

•• cpdisk

make exact copies of a removable disk

This procedure copies the contents of a removable disk into the machine and then allows the user to make exact copies of it. These copies are identical to the original in every way. The copies are made by first reading the original removable disk entirely into the machine and then writing it out onto duplicate disks. The procedure will fail if there is not enough space in the system to hold the original disk.

••• erase

erase data from removable disk

This procedure erases a removable disk by overwriting it with null bytes. The main purpose is to remove data that the user does not want seen. Once performed, this operation is irreversible.

••• format

format new removable disks

Format prepares new removable disks for use. Once formatted, programs and data can be written on the disks.

••• harddisk

hard disk management menu

The subcommands in this menu provide functions for using hard disks. For each hard disk, the disk can be partitioned with default partitioning or the current disk partitioning can be displayed.

•••• display

display hard disk partitioning

Display will allow the user to display the hard disk partitioning. This will inform the user of current disk partitioning information.

•••• partitioning

partition a hard disk

Partitioning configures hard disks. This will allow you to partition a hard disk according to the default partitioning.

••• makefsys

create a new file system on a removable disk

Makefsys creates a new file system on a removable disk which can then store data which the user does not wish to keep on the hard disk. When "mounted", the file system has all the properties of a file kept on the hard disk, except that it is smaller.

••• mountfsys

mount a removable disk file system

Mountfsys mounts a file system, found on a removable disk, making it

available to the user. The file system is unmounted with the "umountfsys" command. THE DISK MUST NOT BE REMOVED WHILE THE FILE SYSTEM IS STILL MOUNTED.

•• **umountfsys**

unmount a removable disk file system

Umountfsys unmounts a file system, allowing the user to remove the disk. THE DISK MUST NOT BE REMOVED UNTIL THE FILE SYSTEM IS UNMOUNTED.

• **filemgmt**

file management menu

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them onto diskettes and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine which files might be best kept on diskette based on age or size.

•• **backup**

backup files from integral hard disk to removable disk or tape

Backup saves copies of files from the integral hard disk file systems to removable disk or tape. There are two kinds of backups:

COMPLETE — copies all files (useful in case of serious file system damage)

INCOMPLETE — copies files changed since the last backup

The normal usage is to do a complete backup of each file system and then periodically do incremental backups. 2 cycles is recommended (one set of complete backups and several incrementals to each cycle). Files backed up with "backup" are restored using "restore".

•• **diskuse**

display how much of the hard disk is being used

Diskuse lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

•• **fileage**

list files older than a particular date

Fileage prints the names of all files older than the date specified by the user. If no date is entered, all files older than 90 days will be listed. If no directory is specified to look in, the user HOME directory will be used.

•• **filesize**

list the largest files in a particular directory

Filesize prints the names of the largest files in a specific directory. If no directory is specified, the users HOME directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

•• **restore**

restore files from "backup" and "store" media to integral hard disk

Restore copies files from disks and tapes made by "backup" and "store"

back onto the hard disk. You can restore individual files, directories of files, or the entire contents of a disk or tape. The user can restore from both "incremental" and "complete" media. The user can also list the names of files stored on the disk or tape.

•• store

store files and directories of files onto disk or tape

Store copies files from the integral hard disk to disk or tape and allows the user to optionally verify that they worked and to optionally remove them when done. Typically, these would be files that the user wants to archive or restrict access to. The user can store single files and directories of files. Use the "restore" command to put stored files back on the integral hard disk and to list the files stored.

•• machinemgmt

machine management menu

Machine management functions are tools used to operate the machine, e.g., turn it off, reboot, or go to the firmware monitor.

•• firmware

stop all running programs then enter firmware mode

This procedure will stop all running programs, close any open files, write out information to the disk (such as directory information), then enter the firmware mode. (Machine diagnostics and other special functions that are not available on the UNIX system.)

•• floppykey

create a "floppy key" removable disk

The "floppy key" removable disk allows the user to enter firmware mode if the firmware password has been changed and then forgotten. Thus the "floppy key" is just that, the "key" to the system and should be protected as such.

• powerdown

stop all running programs, then turn off the machine

Powerdown will stop all running programs, close any open files, write out information to disk (such as directory information), then turn the machine power off.

•• reboot

stop all running programs then reboot the machine

Reboot will stop all running programs, close any open files, write out information to disk (such as directory information), then reboot the machine. This can be used to get out of some types of system trouble, such as when a process cannot be killed.

•• whoson

print list of users currently logged onto the system

Whoson prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

• packagemgmt

package management

These submenus and subcommands manage various software and hardware packages that you install on your machine. Not all optional packages add subcommands here.

- **softwaremgmt**
software management menu

These subcommands permit the user to install new software, remove software, and run software directly from the removable disk it is delivered on. The "remove" and "run" capabilities are dependent on the particular software packages. See the instructions delivered with each package.

- ● **installpkg**
install new software package onto integral hard disk

Install copies files from removable disk onto the integral hard disk and performs additional work if necessary so that the software can be run. From then on, the user will have access to those commands.

- ● **removepkg**
remove previously installed package from integral hard disk

This subcommand displays a list of currently installed optional software packages. Actions necessary to remove the software packages specified by the user will then be performed. The removable disk used to "installpkg" the software is needed to remove it.

- ● **runpkg**
run software package without installing it

This package allows the user to run software from a removable disk without installing it permanently on the system. This is useful if the user does not use the software often or does not have enough room on the system. WARNING: Not all software packages have the ability to run their contents this way. See the instructions that come with the software package.

- **syssetup**
system setup menu

System setup routines allow the user to tell the computer what its environment looks like: what the date, time, and time zone is, what administration and system capabilities are to be under password control, what the machine's name is, etc. The first-time setup sequence is also here.

- ● **admpasswd**
assign or change administrative passwords

Admpasswd lets you set or make changes to passwords for administrative commands and logins such as setup and sysadm.

- ● **datetime**
set the date, time, time zone, and daylight savings time

Datetime tells the computer the date, time, time zone, and whether you observe Daylight Savings Time (DST). It is normally run once when the machine is first set up. If you observe DST, the computer will automatically start to observe it in the spring and return to Standard Time in the fall. The machine has to be turned off and turned back on

again to guarantee that ALL times will be reported correctly. Most are correct the next time the user logs in.

•• nodename

set the node name of this machine

This allows you to change the "node name" of this machine. The "node name" used by various communications networks to identify this machine.

•• setup

set up your machine the very first time

Setup allows the user to define the first login, to set the passwords on the user-definable administration logins and to set the time zone for your location.

•• syspasswd

assign system passwords

Syspasswd lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the "root" login.

• usermgmt

user management menu

These subcommands allow you to add, modify and delete the list of users that have access to your machine. You can also place them in separate groups so that they can share access to files within the group but protect themselves from other groups.

•• addgroup

add a group to the system

Addgroup adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

•• adduser

add a user to the system

Adduser installs a new login ID on the machine. You are asked a series of questions about the user and then the new entry is made. You can enter more than one user at a time. Once this procedure is finished, the new login ID is available.

•• delgroup

delete a group from the system

Delgroup allows you to remove groups from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

•• deluser

delete a user from the system

Deluser allows you to remove users from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the /etc/passwd file.

- lsgroup
 - list groups in the system

Lsgroup will list all the groups that have been entered into the computer using the "addgroup" command. This list is updated automatically by "addgroup" and "delgroup"

- lsuser
 - list users in the system

Lsuser will list all the users that have been entered into the computer using the "adduser" command. This list is updated automatically by "adduser" and "deluser".

- modadduser
 - modify defaults used by adduser

Modadduser allows the user to change some of the defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

- modgroup
 - make changes to a group on the system <not available>

Modgroup allows the user to change all the information about a group that the user enters when "addgroup" is run to set up new groups.

- moduser
 - menu of commands to modify a user's login

This menu contains commands that modify the various aspects of a user's login.

- chgloginid
 - change a user's login ID

This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

- chgpasswd
 - change a user's passwd

This procedure allows removal or change of a user's password. Administrative and system login passwords cannot be changed. To change administrative and system login passwords, see the system setup menu: sysadm syssetup.

- chgshell
 - change a user's login shell

This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

- ttymgmt
 - terminal management

This procedure allows the user to manage the computer's terminal functions.

- baud
 - change the baud rate on a tty line

Baud allows a user to change the baud rate for a tty line and turn on a *getty*(1M) for that line thereby enabling its *login*(1) use. *Baud* displays a list of acceptable values and prompts the user to enter a tty line and the baud rate desired.

•• disable

turn off a tty line

Disable allows a user to turn off a *getty*(1M) for a tty line thereby disabling its *login*(1) use. *Disable* displays a list of acceptable values and prompts the user to enter the tty line desired.

•• enable

turn on a tty line

Enable allows a user to spawn a *getty*(1M) for a tty line thereby enabling that line for *login*(1) use. *Enable* displays a list of acceptable values and prompts the user to enter the tty line desired.

EXAMPLES

sysadm adduser

FILES

The files that support *sysadm* are found in /usr/admin.

The menu starts in directory /usr/admin/menu.

SEE ALSO

mkmenus(1).

NAME

tabs — set tabs on a terminal

SYNOPSIS

tabs [tabspec] [+mn] [-Ttype]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings. The first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is **-8**, i.e., UNIX system "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

- a** 1,10,16,36,72
Assembler, IBM S/370, first format
- a2** 1,10,16,40,72
Assembler, IBM S/370, second format
- c** 1,8,12,16,20,55
COBOL, normal format
- c2** 1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:
<:t-c2 m6 s66 d:>

- c3** 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **-c2**. This is the recommended format for COBOL. The appropriate format specification is:
<:t-c3 m6 s66 d:>

- f** 1,7,11,15,19,23
FORTRAN
- p** 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I
- s** 1,10,55
SNOBOL
- u** 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

- n** A repetitive specification requests tabs at columns $1+n$, $1+2*n$, etc. Note that such a setting leaves a left margin of n columns on TermiNet terminals *only*. Of particular importance is the value **-8**: this represents the UNIX system "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is

the value **-0**, implying no tabs at all.

n1,n2,... The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

--file If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr(1)* command:

tabs -- file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-Ttype *Tabs* usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term(5)*. If no **-T** flag is supplied, *tabs* searches for the \$TERM value in the *environment* (see *environ(5)*). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+mn The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

<i>illegal tabs</i>	when arbitrary tabs are ordered incorrectly.
<i>illegal increment</i>	when a zero or missing increment is found in an arbitrary specification.
<i>unknown tab code</i>	when a "canned" code cannot be found.
<i>can't open</i>	if --file option used, and file can't be opened.
<i>file indirection</i>	if --file option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

pr(1),
environ(5), *term(5)* in the *3B2 Computer System Programmer Reference Manual*.

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

NAME

`tail` — deliver the last part of a file

SYNOPSIS

`tail [±[number][lbc[f]]] [file]`

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **-f** (“follow”) option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail -f fred
```

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail -15cf fred
```

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

`dd(1M)` in the *3B2 Computer System Administration Utilities Guide*.

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

NAME

tee — pipe fitting

SYNOPSIS

tee [**-i**] [**-a**] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. The

-i ignore interrupts;

-a causes the output to be appended to the *files* rather than overwriting them.

NAME

test — condition evaluation command

SYNOPSIS

```
test expr
[expr ]
```

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

- r file** true if *file* exists and is readable.
- w file** true if *file* exists and is writable.
- x file** true if *file* exists and is executable.
- f file** true if *file* exists and is a regular file.
- d file** true if *file* exists and is a directory.
- c file** true if *file* exists and is a character special file.
- b file** true if *file* exists and is a block special file.
- p file** true if *file* exists and is a named pipe (fifo).
- u file** true if *file* exists and its set-user-ID bit is set.
- g file** true if *file* exists and its set-group-ID bit is set.
- k file** true if *file* exists and its sticky bit is set.
- s file** true if *file* exists and has a size greater than zero.
- t [fildes]** true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.
- z s1** true if the length of string *s1* is zero.
- n s1** true if the length of the string *s1* is non-zero.
- s1 = s2** true if strings *s1* and *s2* are identical.
- s1 != s2** true if strings *s1* and *s2* are *not* identical.
- s1** true if *s1* is *not* the null string.
- n1 -eq n2** true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **-ne**, **-gt**, **-ge**, **-lt**, and **-le** may be used in place of **-eq**.

These primaries may be combined with the following operators:

- !** unary negation operator.
- a** binary *and* operator.
- o** binary *or* operator (**-a** has higher precedence than **-o**).
- (expr)** parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

`find(1)`, `sh(1)`.

WARNING

In the second form of the command (i.e., the one that uses `[]`, rather than the word `test`), the square brackets must be delimited by blanks.

NAME

time — time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

times(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

touch — update access and modification times of a file

SYNOPSIS

touch [**-amc**] [mmddhhmm[yy]] files

DESCRIPTION

Touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date(1)*) the current time is used. The **-a** and **-m** options cause *touch* to update only the access or modification times respectively (default is **-am**). The **-c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1).

utime(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

tput — query terminfo database

SYNOPSIS

tput [-Ttype] capname

DESCRIPTION

Tput uses the *terminfo* database to make terminal-dependent capabilities and information available to the shell. *Tput* outputs a string if the attribute (**capability name**) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (0 for TRUE, 1 for FALSE), and does no output.

-Ttype indicates the type of terminal. Normally this flag is unnecessary, as the default is taken from the environment variable **\$TERM**.

Capname indicates the attribute from the *terminfo* database. See *3B2 Computer System Terminal Information Guide*.

EXAMPLES

tput clear	Echo clear-screen sequence for the current terminal.
tput cols	Print the number of columns for the current terminal.
tput -T450 cols	Print the number of columns for the 450 terminal.
bold='tput smso'	Set shell variable "bold" to stand-out mode sequence for current terminal. This might be followed by a prompt: echo "\${bold}Please type in your name: \c"
tput hc	Set exit code to indicate if current terminal is a hardcopy terminal.

FILES

/etc/term/*/*	Terminal descriptor files
/usr/include/term.h	Definition files
/usr/include/curses.h	

SEE ALSO

stty(1).

3B2 Computer System Terminal Information Utilities Guide.

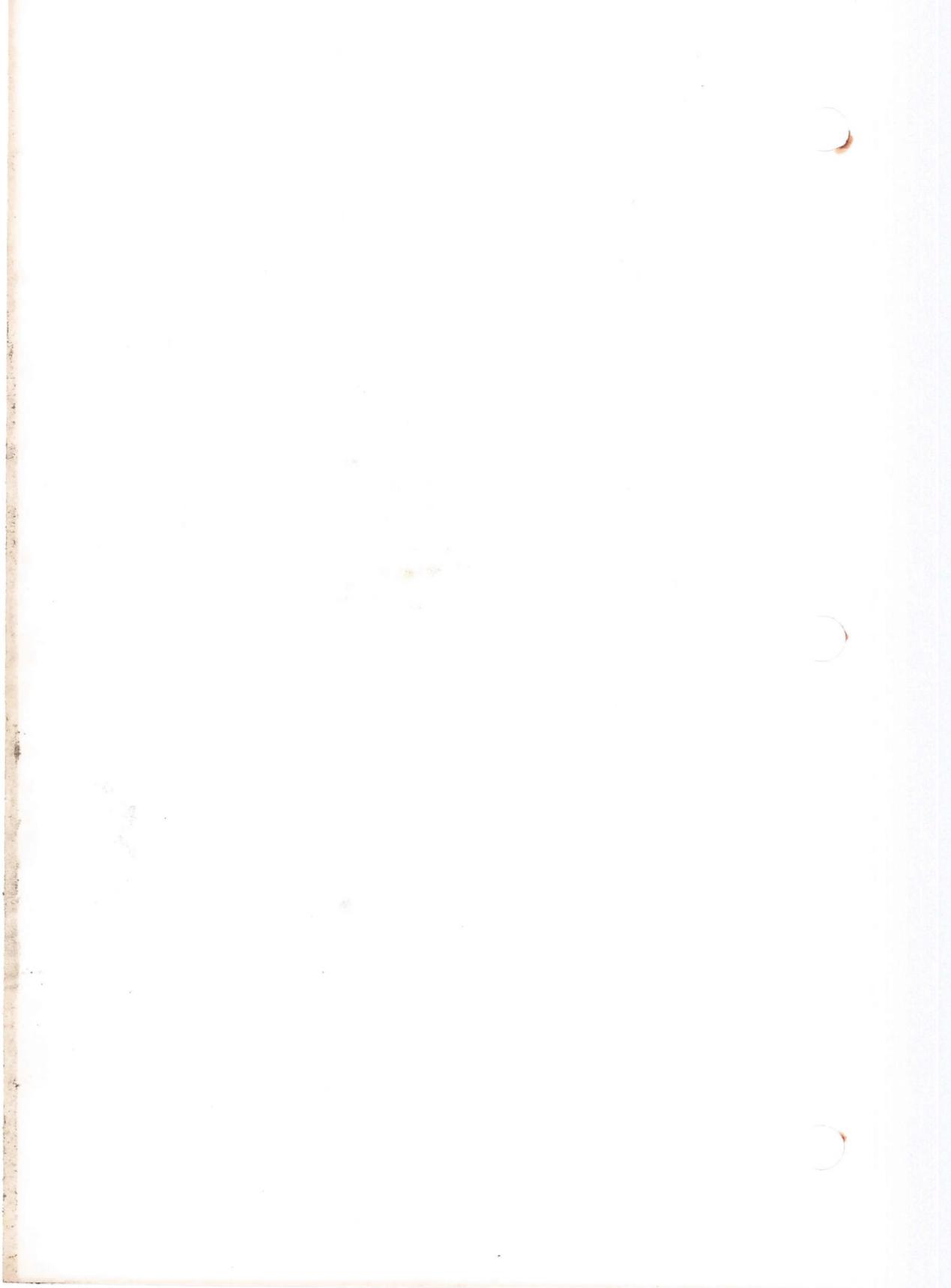
3B2 Computer System Programmer Reference Manual.

DIAGNOSTICS

Tput prints error messages and returns the following error codes on error:

-1	Usage error.
-2	Bad terminal type.
-3	Bad capname.

In addition, if a numeric capname is requested for a terminal that has no value for that capname (e.g., **tput -T450 lines**), **-1** is printed. However, no error codes are returned for string capnames with no values.



NAME

tr — translate characters

SYNOPSIS

tr [-cds] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **-cds** may be used:

- c** Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.
- d** Deletes all input characters in *string1*.
- s** Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

- [a-z]** Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.
- [a*n]** Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character **** may be used as in the shell to remove special meaning from any character in a string. In addition, **** followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabets. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), *sh(1)*.

ascii(5) in the *3B2 Computer System Programmer Reference Manual*.

BUGS

Will not handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

NAME

true, false — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh(1)* such as:

```
while true
do
    command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tty — get the name of the terminal

SYNOPSIS

tty [-l] [-s]

DESCRIPTION

Tty prints the path name of the user's terminal.

- l** prints the synchronous line number to which the user's terminal is connected, if it is on an active synchronous line.
- s** inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES

- 2 if invalid options were specified,
- 0 if standard input is a terminal,
- 1 otherwise.

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and **-l** is specified.

"not a tty" if the standard input is not a terminal and **-s** is not specified.

NAME

umask — set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod(2)* and *umask(2)*). The value of each specified digit is subtracted from the corresponding “digit” specified by the system for the creation of a file (see *creat(2)*). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

Umask can be included in the user's **.profile** and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), *sh(1)*.

chmod(2), *creat(2)*, *umask(2)* in the *3B2 Computer System Programmer Reference Manual*.

NAME

uname — print name of current UNIX system

SYNOPSIS

```
uname [ -snrvma ]
uname [ -S system name ]
```

DESCRIPTION

Uname prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by *uname*(2) to be printed:

- s** print the system name (default).
- n** print the nodename (the nodename may be a name that the system is known by to a communications network).
- r** print the operating system release.
- v** print the operating system version.
- m** print the machine hardware name.
- a** print all the above information.

On the 3B2 computer, the system name and the nodename may be changed by specifying a system name argument to the **-S** option. The system name argument is restricted to 8 characters. Once executed, the system name and the nodename will be changed to the argument specified by the user. Only the super-user is allowed this capacity.

SEE ALSO

uname(2) in the *3B2 Computer System Programmer Reference Manual*.

NAME

uniq — report repeated lines in a file

SYNOPSIS

uniq [**-ude** [**+n**] [**-n**]] [**input** [**output**]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort(1)*. If the **-u** flag is used, just the lines that are not repeated in the original file are output. The **-d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **-u** and **-d** mode outputs.

The **-c** option supersedes **-u** and **-d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

-n The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+n The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), *sort(1)*.

NAME

units — conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch

You want: cm

* 2.540000e+00
/ 3.937008e-01

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 lbs force/in²

You want: atm

* 1.020689e+00
/ 9.797299e-01

Units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi ratio of circumference to diameter,
c speed of light,
e charge on an electron,
g acceleration of gravity,
force same as g,
mole Avogadro's number,
water pressure head per unit height of water,
au astronomical unit.

Pound is not recognized as a unit of mass; lb is. Compound names are run together, (e.g., lightyear). British units that differ from their U.S. counterparts are prefixed thus: brgallon. For a complete list of units, type:

cat /usr/lib/unittab

FILES

/usr/lib/unittab

NAME

usage — retrieve a command description and usage examples

SYNOPSIS

[**help**] **usage** [**-d**] [**-e**] [**-o**] [**command_name**]

DESCRIPTION

The UNIX System *help* Facility command *usage* retrieves information about UNIX system commands. With no argument, *usage* displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by *usage*. The user may also exit to the shell by typing q (for "quit").

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to *usage*. To receive information on the command's description, examples, or options, the user may use the **-d**, **-e**, or **-o** options respectively. (The default option is **-d**.)

From any screen in the facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed was entered at a first level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the *help* facility scrolls the data that is presented to the user. If a user prefers to have the screen cleared before printing the data (non-scrolling), a variable must be defined in the user's .profile file called SCROLL. The variable SCROLL must be set to no and exported for non-scrolling to occur. If the user later decides that scrolling is desired, the variable SCROLL must be set to yes or deleted from the user's .profile file.

Further information on the UNIX System *help* Facility can be found on the *help*(1), *locate*(1), *starter*(1), and *glossary*(1) manual pages.

SEE ALSO

glossary(1), *help*(1), *locate*(1), *sh*(1), *starter*(1).

term(5) in the 3B2 Computer System Programmer Reference Manual.

WARNINGS

If the TERM variable is not set in the user's .profile file, then TERM will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5). The *help* facility assumes that tabs are set on the user's terminal.

NAME

vi — screen-oriented (visual) display editor based on *ex*

SYNOPSIS

```
vi [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ +command ] name ...
view [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ +command ] name ...
vedit [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ +command ] name ...
```

DESCRIPTION

Vi (visual) is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by *vi*:

- t tag** Edit the file containing the *tag* and position the editor at its definition.
- rfile** Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- wn** Set the default window size to *n*. This is useful when using the editor over a slow speed line.
- R** Read only mode; the **readonly** flag is set, preventing accidental overwriting of the file.
- +command** The specified *ex* command is interpreted before editing begins.

The *name* argument indicates files to be edited.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. The **report** flag is set to 1, and the **showmode** and **novice** flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command	Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
Input	Entered by the following options a i A I o O c C s S R . Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line	Reading input for : / ? or !; terminate with CR to execute, interrupt to cancel.

COMMAND SUMMARY**Sample commands**

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i text ESC	insert text <i>abc</i>
c w <i>new</i> ESC	change word to <i>new</i>
e a ESC	pluralize word
x	delete a character
d w	delete a word
d d	delete a line
3 d d	... 3 lines

u	undo previous change
ZZ	exit vi, saving changes
:q!CR	quit, discarding changes
/textCR	search for <i>text</i>
^U ^D	scroll up or down
:ex cmdCR	any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.

line/column number	z G
scroll amount	^D ^U
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete cmd
^?	(delete or rubout) interrupts
^L	reprint screen if ^? scrambles it
^R	reprint screen if ^L is → key

File manipulation

:wCR	write back changes
:qCR	quit
:q!CR	quit, discard changes
:e nameCR	edit file <i>name</i>
:e!CR	redit, discard changes
:e + nameCR	edit, starting at end
:e +nCR	edit starting at line <i>n</i>
:e #CR	edit alternate file
	synonym for :e #
:w nameCR	write file <i>name</i>
:w! nameCR	overwrite file <i>name</i>
:shCR	run shell, then return
:cmdCR	run <i>cmd</i> , then return
:nCR	edit next file in arglist
:n argsCR	specify new arglist
^G	show current file and line
:ta tagCR	to tag file entry <i>tag</i>
^]	:ta, following word is <i>tag</i>

In general, any *ex* or *ed* command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a CR.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
G	go to specified line (end default)
/pat	next line matching <i>pat</i>
?pat	prev line matching <i>pat</i>
n	repeat last / or ?
N	reverse last / or ?
/pat/+n	<i>n</i> th line after <i>pat</i>
?pat?-n	<i>n</i> th line before <i>pat</i>
 	next section/function
 	previous section/function
(beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching () { or }

Adjusting the screen

^L	clear and redraw
^R	retry, eliminate @ lines
zCR	redraw, current at window top
z-CR	... at bottom
z.CR	... at center
/pat/z-CR	<i>pat</i> line at bottom
zn.CR	use <i>n</i> line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

"	move cursor to previous context
"	... at first non-white in line
mx	mark current position with letter <i>x</i>
'x	move cursor to mark <i>x</i>
'x	... at first non-white in line

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
j	next line, same column
k	previous line, same column

Character positioning

^	first non white
0	beginning of line
\$	end of line
h or →	forward
l or ←	backwards
^H	same as ←
space	same as →
fx	find <i>x</i> forward
Fx	f backward
tx	upto <i>x</i> forward
Tx	back upto <i>x</i>
;	repeat last f F t or T
,	inverse of ;
	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Corrections during insert

^H	erase last character
^W	erase last word
erase	your erase, same as ^H
kill	your kill, erase input this line
\	quotes ^H, your erase and kill
ESC	ends insertion, back to command
^?	interrupt, terminates insert
^D	backtab over <i>autoindent</i>
↑^D	kill <i>autoindent</i> , save for next
0^D	... but at margin next also
^V	quote non-printing character

Insert and replace

a	append after cursor
i	insert before cursor
A	append at end of line
I	insert before first non-blank
o	open line below
O	open above
rx	replace single char with <i>x</i>
R <i>text</i> ESC	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since w moves over a word, dw deletes the word that would be moved over. Double the operator, e.g., dd to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command
=	indent for LISP

Miscellaneous Operations

C	change rest of line (c\$)
D	delete rest of line (d\$)
s	substitute chars (cl)
S	substitute lines (cc)
J	join lines
x	delete characters (dl)
X	... before cursor (dh)
Y	yank lines (yy)

Yank and Put

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

p	put back text after cursor
P	put before cursor
"xp	put from buffer x
"xy	yank to buffer x
"xd	delete into buffer x

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"dp	retrieve d'th last delete

AUTHOR

Vi and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1).
3B2 Computer System Editing Utilities Guide.

BUGS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

wait — await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with & have completed, and report on abnormal terminations.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.



NAME

wall — write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

Wall reads its standard input until an end-of-file. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically prior to shutting down the system.

The sender must be super-user to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/tty*

SEE ALSO

mesg(1), *write*(1).

DIAGNOSTICS

“Cannot send to ...” when the open on a user’s tty file fails.

NAME

wc — word count

SYNOPSIS

 wc [-lwc] [names]

DESCRIPTION

Wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **-lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

who — who is on the system

SYNOPSIS

who [-uTHqpdbrtas] [file]

who am i

who am I

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the /etc/utmp file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be /etc/wtmp, which contains a history of all the logins since the file was last created.

Who with the *am i* or *am I* option identifies the invoking user.

Except for the default *-s* option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

- u** This option lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory /dev. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in /etc/inittab (see *inittab(4)*). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.
- T** This option is the same as the **-u** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. Root can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed.
- I** This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field does not exist.
- H** This option will print column headings above the regular output.
- q** This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.
- p** This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in /etc/inittab. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from /etc/inittab that spawned this process. See *inittab(4)*.

- d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait(2)*), of the dead process. This can be useful in determining why a process terminated.
- b This option indicates the time and date of the last reboot.
- r This option indicates the current *run-level* of the *init* process.
- t This option indicates the last change to the system clock (via the *date(1)* command) by *root*. See *su(1)*.
- a This option processes */etc/utmp* or the named *file* with all options turned on.
- s This option is the default and lists only the *name*, *line*, and *time* fields.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), *login(1)*, *mesg(1)*.
init(1M), *su(1M)* in the *3B2 Computer System Administration Utilities Guide*.
wait(2), *inittab(4)*, *utmp(4)* in the *3B2 Computer System Programmer Reference Manual*.

NAME

write — write to another user

SYNOPSIS

write *user* [*line*]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from yourname (tty??) [date]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., *tty00*); otherwise, the first writable instance of the user found in */etc/utmp* is assumed and the following message posted:

user is logged on more than one place.

You are connected to "*terminal*".

Other locations are:

terminal

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, such as *pr(1)* disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character ! is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., (o) for "over") so that the other person knows when to reply. The signal (oo) (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp to find user
/bin/sh to execute !

SEE ALSO

mail(1), *mesg(1)*, *pr(1)*, *sh(1)*, *who(1)*.

DIAGNOSTICS

"*user is not logged on*" if the person you are trying to *write* to is not logged on.

"*Permission denied*" if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*" if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*" if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs — construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

Xargs combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's \$PATH. If *command* is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see **-i** flag). Flags **-i**, **-l**, and **-n** determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **-l** vs. **-n**), the last flag has precedence. *Flag* values are:

-lnumber

Command is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted, 1 is assumed. Option **-x** is forced.

-ireplstr

Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **-x** is also forced. {} is assumed for *replstr* if not specified.

-nnumber

Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option **-x** is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

-t	Trace mode: The <i>command</i> and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
-p	Prompt mode: The user is asked whether to execute <i>command</i> each invocation. Trace mode (-t) is turned on to print the command instance to be executed, followed by a ?... prompt. A reply of y (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of <i>command</i> .
-x	Causes <i>xargs</i> to terminate if any argument list would be greater than <i>size</i> characters; -x is forced by the options -i and -l . When neither of the options -i , -l , or -n are coded, the total length of all arguments must be within the <i>size</i> limit.
-ssize	The maximum total size of each argument list is set to <i>size</i> characters; <i>size</i> must be a positive integer less than or equal to 470. If -s is not coded, 470 is taken as the default. Note that the character count for <i>size</i> includes one extra character for each argument and the count of characters in the command name.
-eofstr	<i>Eofstr</i> is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if -e is not coded. The value -e with no <i>eofstr</i> coded turns off the logical EOF string capability (underbar is taken literally). <i>Xargs</i> reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs will terminate if either it receives a return code of **-1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh(1)*) with an appropriate value to avoid accidentally returning with **-1**.

EXAMPLES

The following will move all files from directory \$1 to directory \$2, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

1. ls | xargs -p -l ar r arch
2. ls | xargs -p -l | xargs ar r arch

The following will execute *diff(1)* with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs -n2 diff
```

SEE ALSO

sh(1).